Cell Broadband Engine

Hardware Initialization Guide

Version 1.3

March 31, 2006

Printed in the United States of America March 2006

IBM Systems and Technology Group
2070 Route 52, Bldg. 330
Hopewell Junction, NY 12533-6351

The IBM home page can be found at **ibm.com**
The IBM Semiconductor Solutions home page can be found at **ibm.com**/chips

BE_HIG_Title.fm.1.3
March 31, 2006

Contents

List of Figures

List of Tables

Preface

This document describes the sequences needed to initialize the Cell Broadband Engine™ (CBE) processor, from the Power-On Reset (POR) sequence through calibration of memory and I/O interfaces and the PPE firmware sequence. The information does not assume any specific system implementation. Some sections of the initialization sequences, such as that would interface to support chip, are system-specific and must be supplied by the system-hardware and system-software designers using this document.

This document is intended for system hardware and software designers who plan to initialize the CBE processor on their own systems. Readers of this manual should be familiar with the documents listed below. Numbers and sample code are examples only, and they may need to be changed, depending on a specific system configuration and chip revision used (see the relevant *Cell Broadband Engine Datasheet*).

The document should provide adequate detail for basic initialization. Additional specific implementation details may be available under non-disclosure agreement from either Sony, Toshiba, IBM, or Rambus. System-specific details should be available from the specific system supplier.

Related Publications

A list of reference materials for the CBE Hardware Initialization Guide follows.

Title	Version	Date
Cell Broadband Engine Architecture	1.0	July 8, 2005
PowerPC User Instruction Set Architecture, Book I	2.02	January 28, 2005
PowerPC Virtual Environment Architecture, Book II	2.02	January 28, 2005
PowerPC Operating Environment Architecture, Book III	2.02	January 28, 2005
PowerPC Microprocessor Family: The Programming Environments for 64- and 32-Bit Microprocessors	2.0	June 10, 2003
Synergistic Processor Unit Instruction Set Architecture	1.1	January 30, 2006
Cell Broadband Engine Registers	3.0	August 31, 2005
Cell Broadband Engine Datasheet[1]	3.36	February 3, 2006
Cell Broadband Engine Programming Handbook	1.0	Forthcoming
Broad Band Processor Pervasive Workbook - Functional,[1]	0.3	June 9, 2005

1. Available under non-disclosure agreement from Sony, Toshiba, or IBM.

I/O Reference Documentation

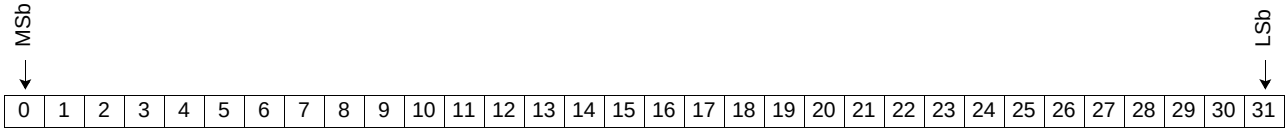
The following Rambus documents are available from Rambus to assist in board design and I/O calibration of the CBE interfaces.

Document Name	Version	Date
Rambus XDR Initialization Guide (DL-0178)	0.93	January 2006
Rambus XDR Architecture (DL-0161)	0.80	March 2004
Rambus BE-XIO Specification (DD2.0) - Addendum to DL-0153 XDR IO Cell Datasheet (DL-187)	0.84.1	September 2004
Rambus XDR I/O Cell (DL-153)[1]	0.84	September 2004
Rambus XDR DRAM 8x4Mx16 (DL-130)	0.90	January 2006
Rambus XDR System Design Guide (DL-0171)	0.80	March 2004
Rambus FlexIO Processor Bus Interface Cell Datasheet (DL-0159)	0.90	September 2005
Rambus BE-FlexIO Processor Bus Interface Cell - Addendum to rev 0.90 FlexIO Processor Bus Interface Cell Datasheet (DL-0159)	0.90.1	September 2005
Rambus STI Specific System Design Guide (DL-0179)	0.80	March 2004
Rambus Redwood System Design Guide (DL-0172)	0.80	March 2004

1. Except when referencing the publication, this is referred to as the XDR I/O cell in this document.

Conventions and Notation

In this document, standard IBM notation is used, meaning that bits and bytes are numbered in ascending order from left to right. Thus, for a 4-byte word, bit 0 is the most-significant bit and bit 31 is the least-significant bit.



Throughout this document, standard IBM big-endian notation is used, meaning that bytes are numbered in ascending order from left to right. Big-endian and little-endian byte ordering are described in the *Cell Broadband Engine Architecture*.

Notation for bit encoding is as follows:

- Hexadecimal values are preceded by x and enclosed in single quotation marks. For example: x‘0A00’.
- Binary values in sentences appear in single quotation marks. For example: ‘1010’.

The following software documentation conventions are used in this manual:

1. Command (or instruction) names are written in **bold** type. For example: **lmw**.
2. Variables are written in italic type. Required parameters are enclosed in angle brackets. Optional parameters are enclosed in brackets. For example: **dcb*[t]*<*s*>*[t]***.

In some cases, registers are referred to by the register mnemonic. Fields are then referred to by the register mnemonic followed by the field name enclosed in brackets. An equal sign followed by a value indicates the value to which the field is set. For example, MFC_SR1[R] = 0). For more information, see *Referencing Registers, Fields, and Bit Ranges*.

The following symbols are used in this document:

&	bitwise AND
|	bitwise OR
%	modulus
=	equal to
! =	not equal to
x ≥	greater than or equal to
x ≤	less than or equal to
x >> y	shift to the right; for example, 6 >> 2 = 1; least-significant y-bits are dropped
x << y	shift to the left; for example, 3 << 2 = 12; least-significant y-bits are replaced zeros

Referencing Registers, Fields, and Bit Ranges

Registers are referenced by their full name or by their short name (also called the register mnemonic). Fields within registers are referenced by their full field name or by their field name. The field name or names are enclosed in brackets [ ].The following table describes how registers, fields, and bit ranges are referenced in this document and provides examples of the references.

Type of Reference	Format	Example
Reference to a specific register and a specific field using the register short name and the field name(s), bit number(s), or bit range.	Register_Short_Name[Bit_FieldName]	MSR[FE0]
	Register_Short_Name[Bit_Number]	MSR[52]
	Register_Short_Name[Field_Name1, Field_Name2]	MSR[FE0, FE1]
	Register_Short_Name[Bit_Number, Bit_Number]	MSR[52, 55]
	Register_Short_Name[Starting_Bit_Number:Ending_Bit_Number]	MSR[39:44]
Reference to a specific register and the setting for a specific field, bit, or range of bits using the register short name and the field name(s), bit number(s), or bit range that is followed by an equal sign (=) and a value that field.	Register_Short_Name[Field_Name] = '*n*' (where *n* is a binary value for the bit or bit range)	MSR[FE0] ='1'
	Register_Short_Name[Field_Name] =x'*n*' (where *n* is a hexadecimal value for the bit or bit range)	MSR[FE] = x'1'
	Register_Short_Name[Bit_Number] = '*n*' (where *n* is a binary value for the bit or bit range)	MSR[52] = '0'
	Register_Short_Name[Bit_Number]] = x'*n*' (where *n* is a hexadecimal value for the bit or bit range)	MSR[52] = x'0'
	Register_Short_Name[Starting_Bit_Number:Ending_Bit_Number] = '*n*' (where *n* is the binary or hex value for the bit or bit range)	MSR[39:43] = '10010' [39:43] = '10010'
	Register_Short_Name[Starting_Bit_Number:Ending_Bit_Number] = '*n*' (where *n* is the hexadecimal value for the field or bits)	MSR[39:43] = x'11' [39:43] = x'11'

Note: The register short name is also called the register mnemonic.

Referencing signal names from the Datasheet

Signal names used from the *Cell Broadband Engine Datasheet* are in uppercase letters (SIGNAL). Active low signals have an overbar over the signal name ($\overline{\text{SIGNAL}}$).

1. Overview of the Cell Broadband Engine™ Processor

This document covers the initialization of the CBE processor, from first applying power to the step just prior to loading a hypervisor or an operating system. It includes the early Power-On Reset (POR) sequence, calibration of memory and I/O interfaces, and the PPE firmware sequence. It is written for system designers and laboratory engineers involved in booting the CBE processor. It does not cover any specific system design. Therefore, the only system requirements covered are those that apply to any system built around the CBE processor. Documents that are frequently referenced are listed in the *Preface* on page 11.

Even though the main focus of this document is on the initialization of the CBE processor, two additional topics are included (as appendices) that apply to normal operation and are not part of the initialization—Livelock Resolution Mode, and Fault Isolation Register (FIR) features.

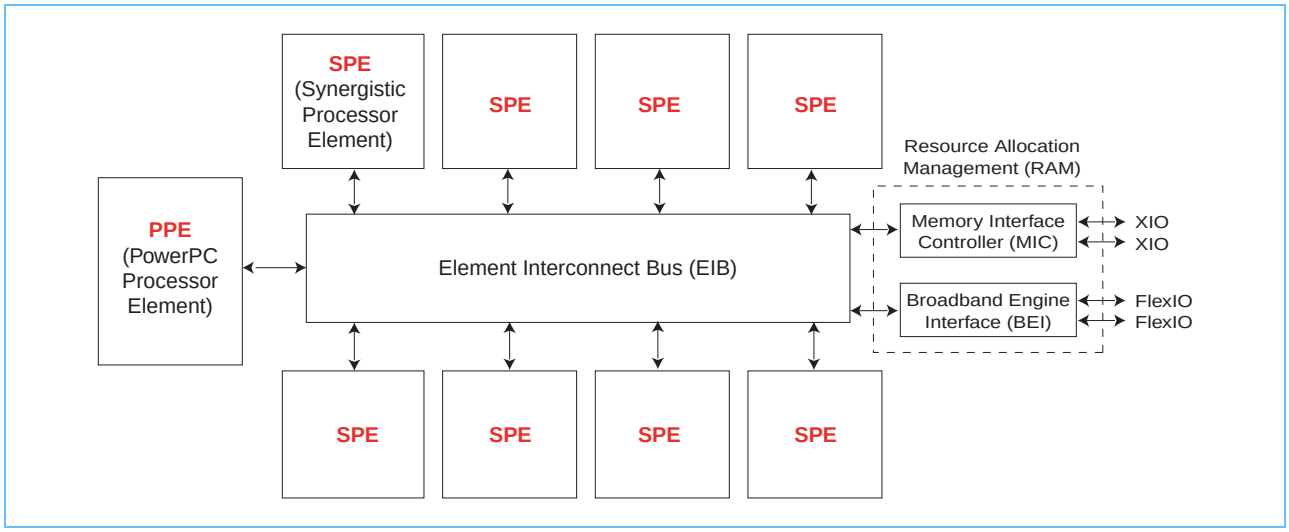
Throughout this document, the early Rambus terms *RRAC* and *YRAC* should be understood as the Rambus *FlexIO* and *XDR I/O (XIO)* interfaces, respectively.

1.1 Hardware Overview

The CBE processor is a single-chip multiprocessor with nine processor elements, plus onchip memory and I/O controllers, operating on a shared, coherent memory. In this respect, it extends current trends in PC and server processors. Although all of the CBE processor's processing elements share main storage, their function is specialized into two types: the PowerPC Processor Element (PPE), and the Synergistic Processor Element (SPE). The CBE processor has one PPE and eight identical SPEs.

All of the CBE processor elements are connected to each other and to external devices by a high-bandwidth, memory-coherent bus, called the Element Interconnect Bus (EIB). *Figure 1-1* on page 15 shows a block diagram of the CBE processor.

Figure 1-1. Cell Broadband Engine Overview



1.1.1 The Processor Elements

There are nine processor elements in the CBE processor—one PowerPC Processor Element (PPE) and eight identical Synergistic Processor Elements (SPEs). All processor elements are connected to each other and to the on-chip memory and I/O controllers by the high-bandwidth, coherent Element Interconnect Bus (EIB).

The PPE is the control processor. It contains a 64-bit, dual-thread PowerPC Architecture™ RISC core with a traditional PowerPC virtual-memory subsystem. It has 32-KB Level-1 (L1) instruction and data caches and a 512-KB Level-2 (L2) unified (instruction and data) cache. It is intended primarily for control processing, running operating systems, managing system resources, and managing SPE threads. It can run legacy PowerPC Architecture software and performs well executing system-control code. The instruction set for the PPE is an extended version of the PowerPC instruction set. It includes the Vector/SIMD Multimedia Extensions and associated C/C++ intrinsic extensions.

The eight SPEs are SIMD processor elements that are optimized for data-rich operations allocated to them by the PPE. Each of these identical elements contains a RISC core, 256-KB software-controlled local store for instructions and data, and a 128-bit, 128-entry unified register file. The SPEs support a special SIMD instruction set—the *Synergistic Processor Unit Instruction Set Architecture*—and a unique set of commands for managing tasks like Direct Memory Access (DMA) transfers between main storage and an SPE's local store and for interprocessor messaging. An SPE relies on DMA transfers to asynchronously move data and instructions between main storage and its local stores while the SPE performs computation in parallel. Each SPE has a PowerPC-architecture-compatible memory-management unit. SPE DMA transfers access main storage using PowerPC effective addresses. As in the PPE, SPE address translation is governed by PowerPC Architecture segment and page tables, which are loaded into the SPEs by privileged software on the PPE. The SPEs are not intended to run a formal operating system.

An SPE communicates with the system by means of its Memory Flow Controller (MFC). An SPE uses a *channel* interface for this communication. SPE channels are unidirectional access ports, between the SPE's execution units and the SPE's MFC, to function-specific registers and queues implemented in and managed by the MFC. The PPE and other devices in the system, including other SPEs, can also access this MFC state through the MFC's Memory-Mapped I/O (MMIO) registers and queues, which are visible in the main-storage address space. The SPE channels and their corresponding MMIO state support functions like DMA control, mailboxes, and signal-notification between all processor elements in the system.

1.1.2 Element Interconnect Bus

The Element Interconnect Bus (EIB) is the communication path for commands and data between all processor elements on the CBE processor and the onchip controllers for memory and I/O. The EIB supports full memory-coherent and symmetric multiprocessor (SMP) operations.

The EIB manages four 16-byte-wide data rings, which connect all units on the chip. Each ring transfers 128-byte at a time. Two rings run clockwise, two counterclockwise. Each unit has one on-ramp and one off-ramp. Units can drive and receive simultaneously. Multiple transfers can be in-process concurrently on each ring.

The EIB's internal maximum bandwidth is 96 bytes per processor cycle, and it can support more than 100 outstanding DMA memory requests between main storage and the SPEs. The EIB does not support any particular quality-of-service (QoS) behavior other than to guarantee forward progress. However, a token manager unit resides in the EIB, and software can use it to regulate the rate at which particular devices are allowed to make EIB command requests.

1.1.3 Memory Interface Controller

The Memory Interface Controller (MIC) provides the interface between the EIB and main storage. It supports one or two Rambus Extreme Data Rate (XDR) memory interfaces (channels), which together support between 64 MB and 64 GB of XDR DRAM memory. The interfaces are often referred to as XDR I/O (XIO) cell interfaces.

Memory accesses on each interface are 1 to 8, 16, 32, 64, or 128 bytes, with coherent memory-ordering. Up to 64 reads and 64 writes can be queued. A token manager provides feedback about queue levels.

The MIC supports multiple software-controlled modes, including fast-path mode (for improved latency when command queues are empty), high-priority read (prioritizes PPE reads and SPE atomic reads in front of all other reads), early read (a read can start before a previous write completes), speculative read[1], and slow mode (power management). The MIC implements a closed-page controller (bank rows are closed after being read, written, or refreshed), memory initialization, memory scrubbing, and auxiliary trace data storage (a debug feature).

The XDR DRAM memory is ECC-protected, with multi-bit error detection and optional single-bit error correction. Additional features include early read (optional), write-masking, initial and periodic timing calibration, dynamic width control, sub-page activation, dynamic clock gating, and 4, 8, or 16 banks.

Table 1-1 lists possible memory capacities for the CBE processor, now and in the future. The 1Gb x 4 XDR might be possible, but contact an XDR vendor for current information.

Table 1-1. CBE System Memory Capacity

Memory Per Channel	Configuration	Number of chips
128 MB	512 Mb x 16 XDR	2 (3 with ECC)
256 MB	512 Mb x 8 XDR	4 (5 with ECC)
512 MB	512 Mb x 4 XDR	8 (9 with ECC)
1 GB	1 Gb x 4 XDR[1]	8 (9 with ECC)
1 GB	Synapse 512 Mb x 8 DDR2 SDRAM	16 (18 with ECC)
2 GB	1 Gb x 2 XDR[1]	16 (18 with ECC)
2 GB	Synapse 512 Mb x 4 DDR2 SDRAM	32 (36 with ECC)
4 GB	Synapse 1 Gb x 4 DDR2 SDRAM	32 (36 with ECC)
8 GB	Synapse 1 Gb x 4 DDR2 SDRAM, Double Rank (11s only)	64 (72 with ECC)

1. Contact an XDR vendor for confirmation of availability.

1. Speculative reads are those in which the MIC attempts to perform the memory access even if it does not know the bus response. They are useful for multi-CBE-processor systems.

1.1.4 Broadband Engine Interface

The Cell Broadband Engine Interface (BEI), shown in *Figure 1-2* on page 20, supports I/O interfacing. It includes a Broadband Interface Controller (BIC), I/O Controller (IOC), and Internal Interrupt Controller (IIC), as defined in the *Cell Broadband Engine Architecture* document. It manages data transfers between the EIB and I/O devices and provides I/O address translation and command processing.

The BEI supports two Rambus FlexIO interfaces (channels). One of the two interfaces (IOIF1) supports only a non-coherent I/O Interface (IOIF) protocol, which is suitable for I/O devices. The other interface (IOIF0, also called BIF/IOIF0) is software-selectable between the non-coherent protocol and the fully coherent Broadband Interface (BIF) protocol—the EIB's native internal protocol—which coherently extends the EIB to another device that can be another CBE processor. Thus, a CBE processor is designed to be ganged coherently with other CBE processors to produce a cluster. The BIF and IOIF protocols are both IBM-proprietary.

The FlexIO interface provides seven transmit bytes and five receive bytes of Rambus FlexIO channel interface. At a 500 MHz clock rate (see the *Cell Broadband Engine Datasheet* for actual clock rates) each differential pair carries 5.0 Gbps of data traffic (2.5 Gbps in half-rate mode) at Differential Rambus Signaling Level (DRSL). Each channel is eight bits wide and has its own differential data clock. At the physical layer, the FlexIO interface performs calibration or I/O calibration during the Power-On Reset (POR) sequence to adjust the driver impedance and output levels and to align the channel's eight data bits with the data clock.

One or more bytes of the FlexIO interface can be bonded to an IOIF-protocol or BIF-protocol interface at POR by means of fields in the CBE Configuration Ring. See the fields in *Table 4-1* on page 120 for setting up the number of BIF/IOIF0 and IOIF1 transmit and receive blocks, and the BIF/IOIF0 coherency mode. Up to two devices can be connected by means of an IOIF0/BIF and IOIF1 interface. IOIF0/BIF can be 1to 5 bytes and IOIF1 can be 1 to 2 bytes. *Table 1-2* and *Table 1-3* show the valid configurations for a receiving (Rx) and transmitting (Tx) device, respectively.

In the system, the FlexIO interface connects to another device with similar logic controlling the interface. The card wiring for the channel must comply with the guidelines in the *Rambus Redwood System Design Guide (DL-0172)*.

Table 1-2. Device Connection Rx (Page 1 of 2)

FlexIO_0	FlexIO_1	FlexIO_2	FlexIO_3	FlexIO_4
Device 0				
Device 0				
Device 0				
Device 0				
Device 0				
Device 0				Device 1
Device 0				Device 1
Device 0				Device 1
Device 0				Device 1
				Device 1

Table 1-2. Device Connection Rx (Page 2 of 2)

FlexIO_0	FlexIO_1	FlexIO_2	FlexIO_3	FlexIO_4
Device 0			Device 1	
Device 0			Device 1	
Device 0			Device 1	
			Device 1	

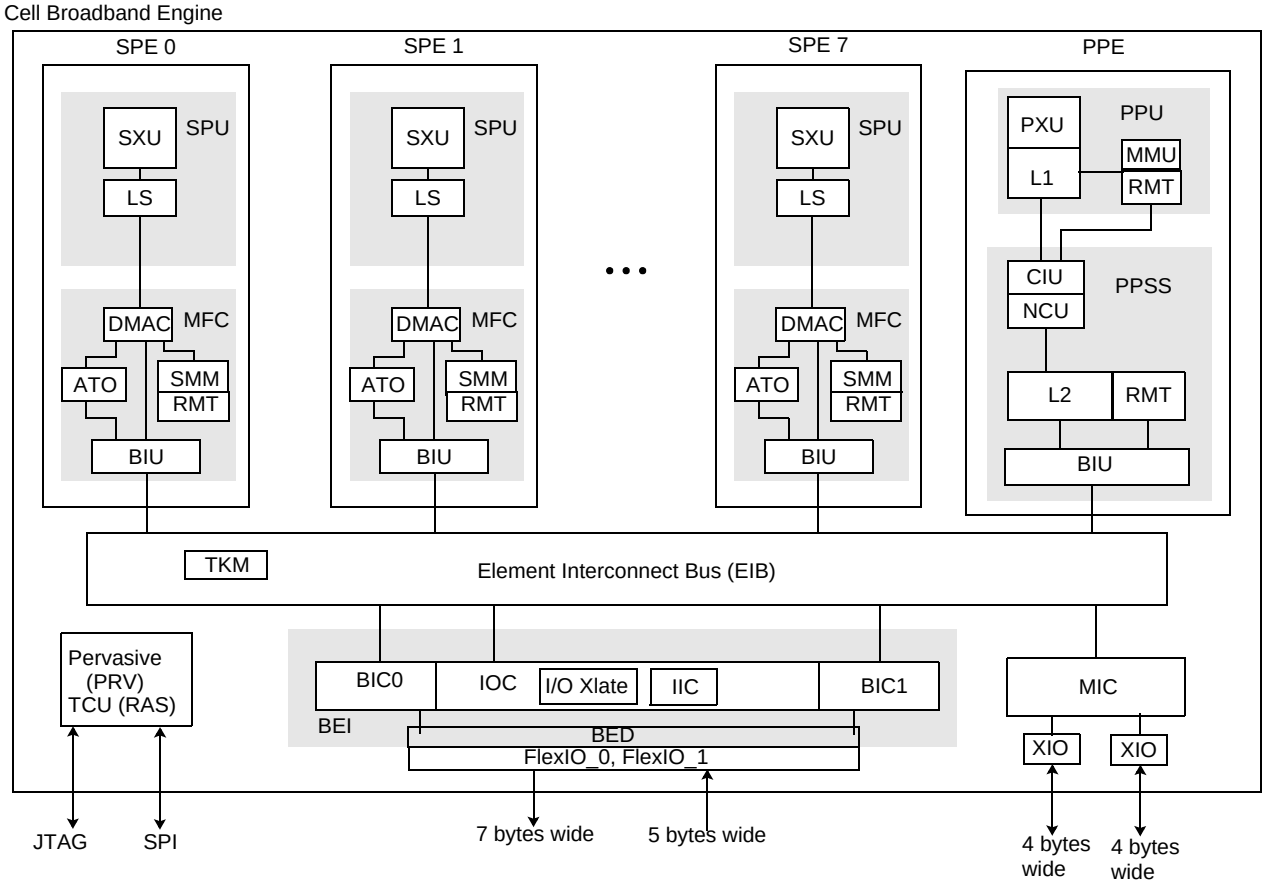
Table 1-3. Device Connection Tx

FlexIO_0	FlexIO_1	FlexIO_2	FlexIO_3	FlexIO_4	FlexIO_5	FlexIO_6
Device 0						
Device 0						
Device 0						
Device 0						
Device 0						
Device 0						
Device 0						Device 1
Device 0						Device 1
Device 0						Device 1
Device 0						Device 1
Device 0						Device 1
Device 0						Device 1
						Device 1
Device 0					Device 1	
Device 0					Device 1	
Device 0					Device 1	
Device 0					Device 1	
Device 0					Device 1	
					Device 1	

1.1.5 Detail Block Diagram

A detailed block diagram of the CBE logic units is shown in *Figure 1-2* on page 20. Most of the unit names in this figure correspond to the unit names used in the *Cell Broadband Engine Registers* document. Some of the names shown in the figure for the SPE are different in the actual logic design; for example, *SMF* is the name used in the *Cell Broadband Engine Registers* document for the MFC, and *SBI* is used for the BIU on the SPE.

Figure 1-2. High-Level Block Diagram of the CBE Processor



ATO	Atomic Unit
BED	CBE Distribution
BEI	Cell Broadband Engine Interface Unit
BIC	Bus Interface Controller
BIU	Bus Interface Unit
CIU	Core Interface Unit
DMAC	Direct Memory Access Controller
EIB	Element Interconnect Bus
FlexIO	Rambus Flexible I/O Cell
IIC	Internal Interrupt Controller
IOC	I/O Interface Controller
I/O Trans	I/O Address Translation
JTAG	IEEE 1149 test access port
L1	Level 1 Cache
L2	Level 2 Cache
LS	Local Store
MFC	Memory Flow Controller
MIC	Memory Interface Controller
MMU	Memory Management Unit
NCU	Noncacheable Unit
PPE	PowerPC Processor Element
PPSS	PowerPC Processor Storage Subsystem
PPU	PowerPC Processor Unit
PRV	Pervasive Logic
PXU	Processor Execution Units (without L1)
RAS	Reliability, Availability, Serviceability
RMT	Replacement Management Table
SMM	Synergistic Memory Management Unit
SPE	Synergistic Processor Element
SPI	Serial Peripheral Interface
SPU	Synergistic Processor Unit
SXU	Synergistic Execution Units (without LS)
TCU	Test Control Unit
TKM	Token Management Unit
XIO	Extreme Data Rate (XDR) I/O Cell

1.2 Clock Domains

The CBE processor has three clock domains, each running asynchronously to the other two domains:

- *CBE Core Clock (NClk)*—This clock times the PPU, SPUs, and parts of the PPSS and MFCs.
- *MIC Clock (MiClk)*—This clock times the Memory Interface Controller (MIC).
- *BIC Core Clock (BClk)*—This clock times the Bus Interface Controller (BIC), which is part of the Broadband Engine Interface (BEI) to the I/O interface.

The following CBE logic blocks run as half the CBE Core Clock frequency (NClk/2):

- Element Interconnect Bus (EIB) and interfaces to the EIB (parts of the PPSS and MFCs).
- I/O Interface Controller (IOC).
- MIC logic that is part of the CBE Core.
- BIC logic that is part of the CBE Core.
- Pervasive logic, which is the logic that provides power management, thermal management, clock control, software-performance monitoring, trace analysis, and so forth.

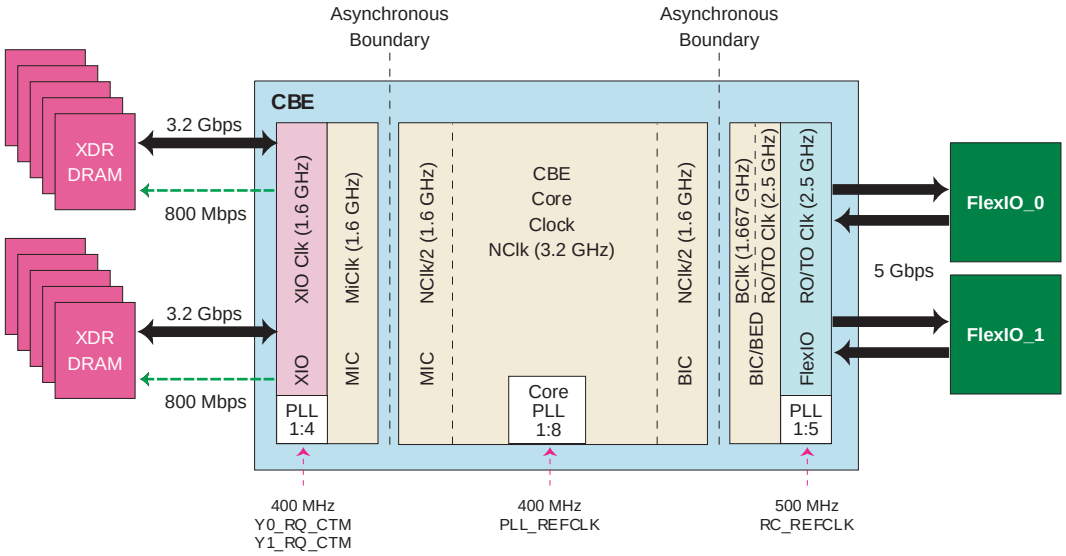
The Rambus XIO cell interfaces run at the XIO clock frequency, and the Rambus FlexIO interfaces run at the receive and transmit clock frequencies (RO Clk and TO Clk).

Figure 1-3 on page 22 shows the clock domains in normal operation. The frequency numbers used in this figure are meant as an example only. For actual frequencies supported on the CBE processor and for specifications for the three Phase Locked Loop (PLL) clock inputs, see the *Cell Broadband Engine Datasheet*.

The following terms are used for the PLL reference clocks and clock multipliers:

- Core PLL Reference Clock (PLL_REFCLK). The multiplier for the PLL_REFCLK is called the Core Clock Multiplier (CCM) in the *Cell Broadband Engine Programming Handbook*.
- XIO PLL Reference Clock per channel (Y0_RQ_CTM, Y1_RQ_CTM).
- FlexIO PLL Reference Clock (RC_REFCLK).

Figure 1-3. CBE Clock Domains in Normal Operation



BClk	BIC Core Clock	NClk	CBE Core Clock
BED	Cell Broadband Engine Distribution Bus	PLL	Phase-Locked Loop
BIC	Bus Interface Controller	PRV	Pervasive Logic Registers
FlexIO	Rambus® FlexIO bus	RO Clk	FlexIO Receive Clock
IOIF	I/O Interface	TO Clk	FlexIO Transmit Clock
MIC	Memory Interface Controller	XDR	Rambus® XDR DRAM
MiClk	MIC Core Clock	XIO	Rambus® XDR I/O Cell

The frequency numbers used in this figure are meant as an example only.

1.3 System Configuration

Figure 1-4 shows a high-level diagram of a basic CBE system that indicates the scope of the material covered in this document. All of the FlexIO calibration code shown in *Section 2.1.5.3 FlexIO Bit and Byte Calibration (I/O Training)* on page 35 is done from CBE processor to CBE processor—that is, it assumes that the IOIF interface has a second CBE processor attached to assist in the calibration. This configuration is used because no a specific IOIF0 or IOIF1 support chip (with their own implementation-specific registers for I/O calibration) is available for this interface testing. By doing CBE processor to CBE processor IOIF initialization, the register references are simplified because they consist only of those registers documented in the *Cell Broadband Engine Registers* document.

Figure 1-4. Basic-System Block Diagram

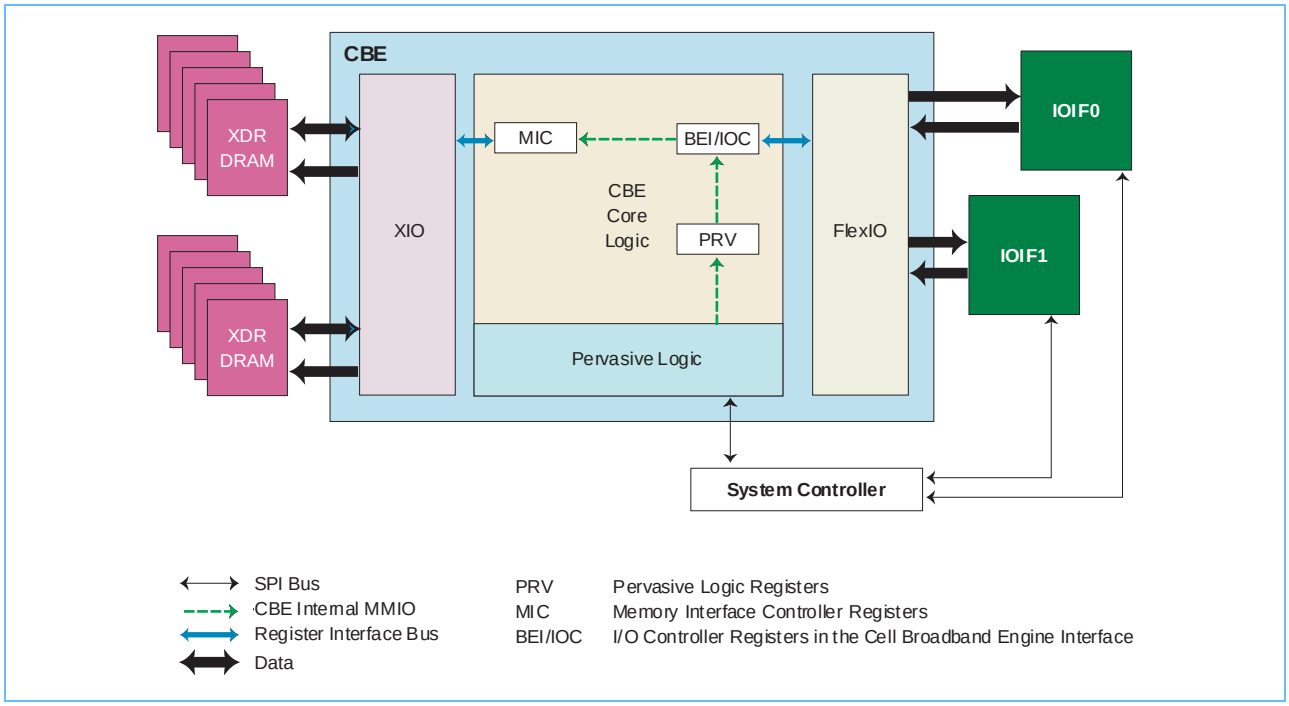
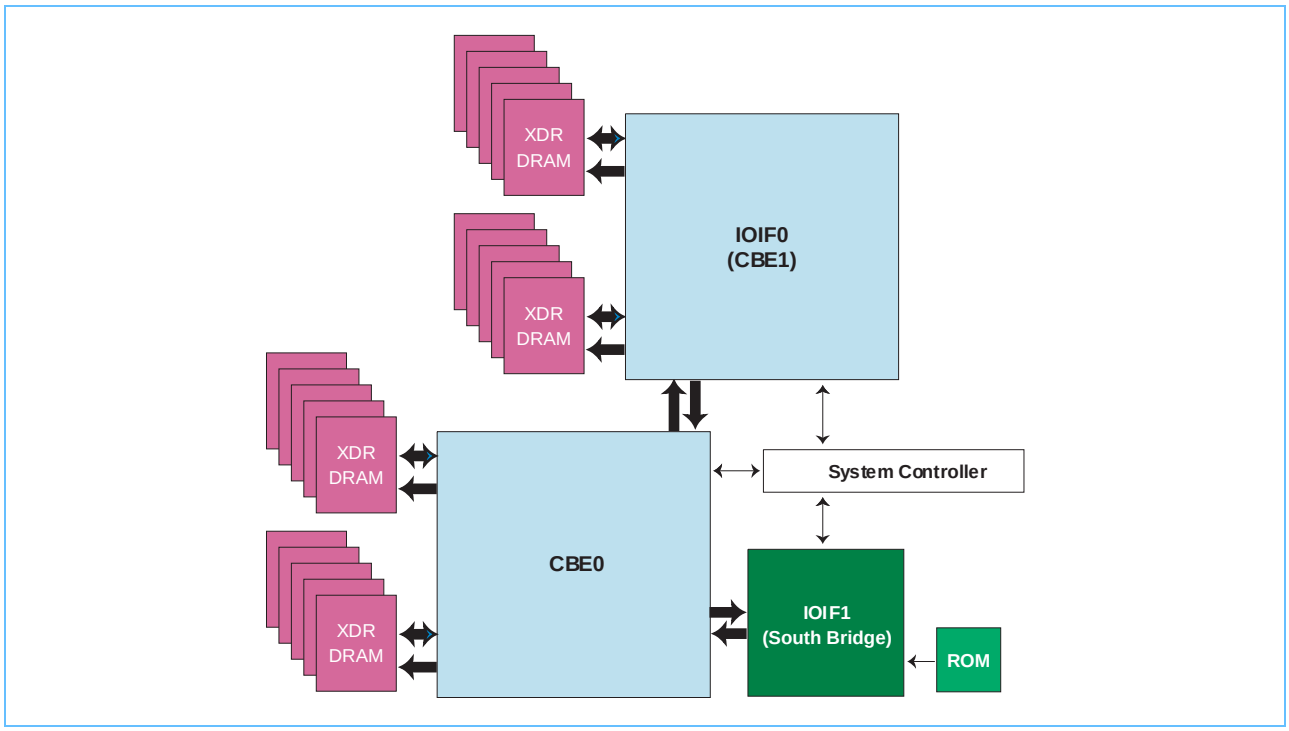


Figure 1-5 shows an example of a larger system than shown in *Figure 1-4*. The system in *Figure 1-5* includes a high-speed chip (such as another CBE processor) connected to the IOIF0 interface, a South Bridge (I/O Bridge) chip connected to the IOIF1 interface, and a ROM chip connected to the South Bridge chip. The South Bridge and ROM chips are not covered in this document. The ROM attached to the South Bridge chip is the same ROM mentioned in *Section 2.2.1 Firmware-Sequence Flowchart and Pseudocode* on page 53.

Figure 1-5. Example Full-System Block Diagram

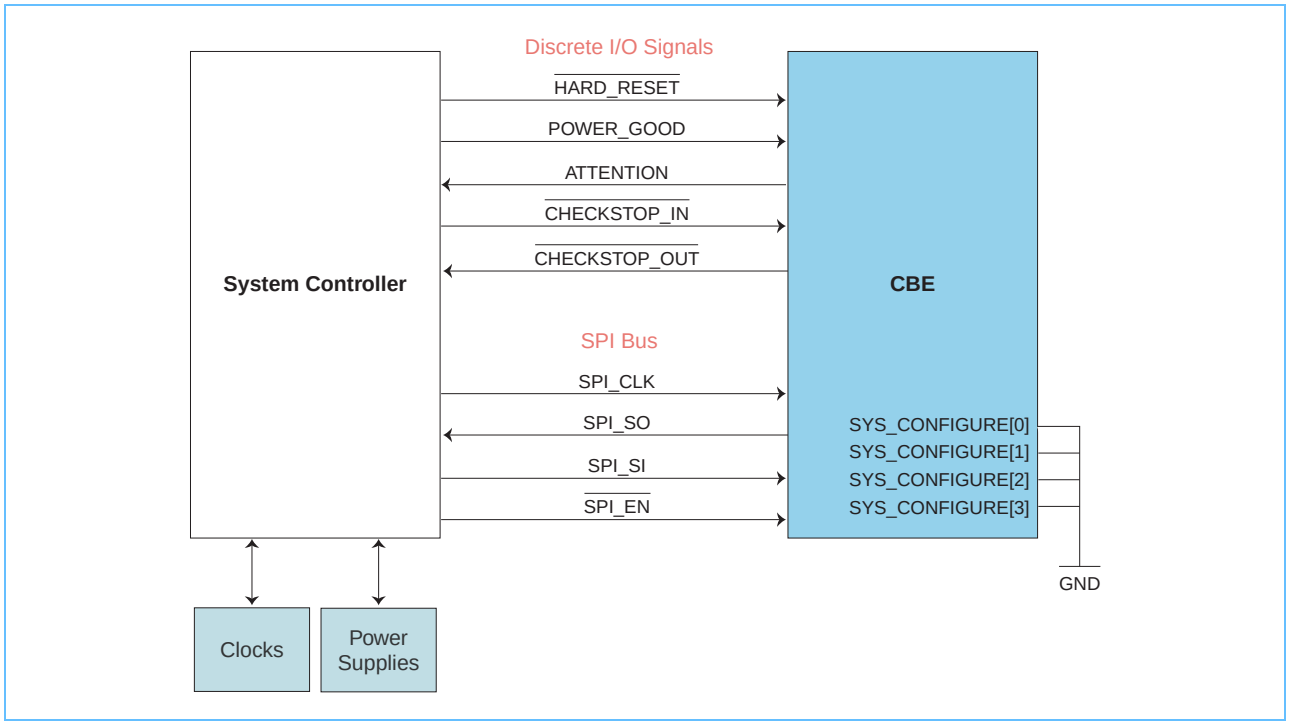


1.4 System Controller Overview

The System Controller is a device external to the CBE processor that participates in the POR sequence prior to the start of firmware code execution. The minimum set of signals that the System Controller must connect to on the CBE processor are shown in *Figure 1-6* on page 25. Because the System Controller is connected to the ATTENTION signal on the CBE processor, it also responds to error conditions on the CBE processor during normal operation.

The System Controller also assists in initializing the support chips on the IOIF0 and IOIF1 interfaces by means of the Serial Peripheral Interface (SPI), and it controls the power supplies, PLLs, and voltage regulator modules (VRMs) on the system board. These System Controller functions must be defined by the system designer and are beyond the scope of this document.

Figure 1-6. Interface Between System Controller and CBE Processor



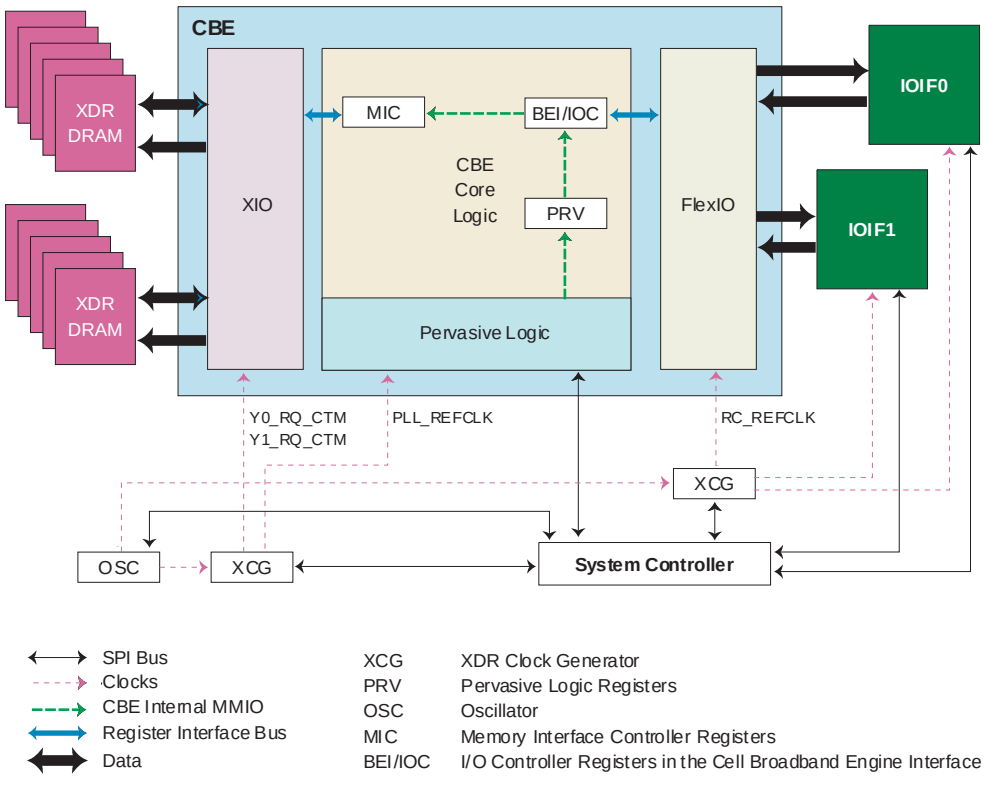
2. Initialization Sequences

This chapter describes the initialization of the CBE processor, from the time that power is applied to the time that the operating system is loaded. The entire initialization is performed in two sequences:

1. *Power-On Reset (POR) Sequence*—This hardware sequence requires the assistance of an external System Controller, and it occurs prior to code executing on the CBE processor. The sequence is divided into three phases, as described in *Section 2.1* and illustrated in *Figure 2-2* on page 29.
2. *Firmware Sequence*—This sequence starts the execution of code on the PPE. It initializes the XIO memory interface, DRAM memory, and some of the PPE hardware-implementation dependent (HID) special-purpose registers (SPRs). The sequence is described in *Section 2.2* on page 52 and illustrated in *Figure 2-6* on page 55.

Figure 2-1 shows a block diagram for a basic system. The Voltage Regulator Module (VRM) is left to the system designer to implement and is therefore not included in this figure. Guidelines for the VRM can be found in the *Cell Broadband Engine Datasheet*. More information about system design and initialization related to the I/O and memory interfaces can be found in the documentation listed in *I/O Reference Documentation* on page 11.

Figure 2-1. Sample CBE System Configuration



2.1 Power-On Reset (POR) Sequence

2.1.1 POR Sequence Summary

The POR sequence is a hardware sequence that relies on handshaking between the CBE processor and the System Controller. The System Controller is responsible for supplying the appropriate data to the CBE processor when requested and in coordinating the initialization of CBE support chips, such as the clock generator and companion chips.

The following major activities are accomplished during the POR sequence:

- Initialize the CBE core logic, including reset the internal state and set the core PLL.
- Adjust the Voltage Regulation Module (VRM) voltage according to Voltage ID (VID) information stored in the CBE processor.
- Load the Configuration-Ring data.
- Calibrate the FlexIO interface (initialization, bit calibration, and byte calibration).
- Initialize the IOIF interface.

Figure 2-2 on page 29 shows a flowchart of the POR sequence. The following sections describe these steps. A summary of the sequence is given in *Section 2.1.1* on page 28.

Figure 2-2. Power-On Reset (POR) Flowchart

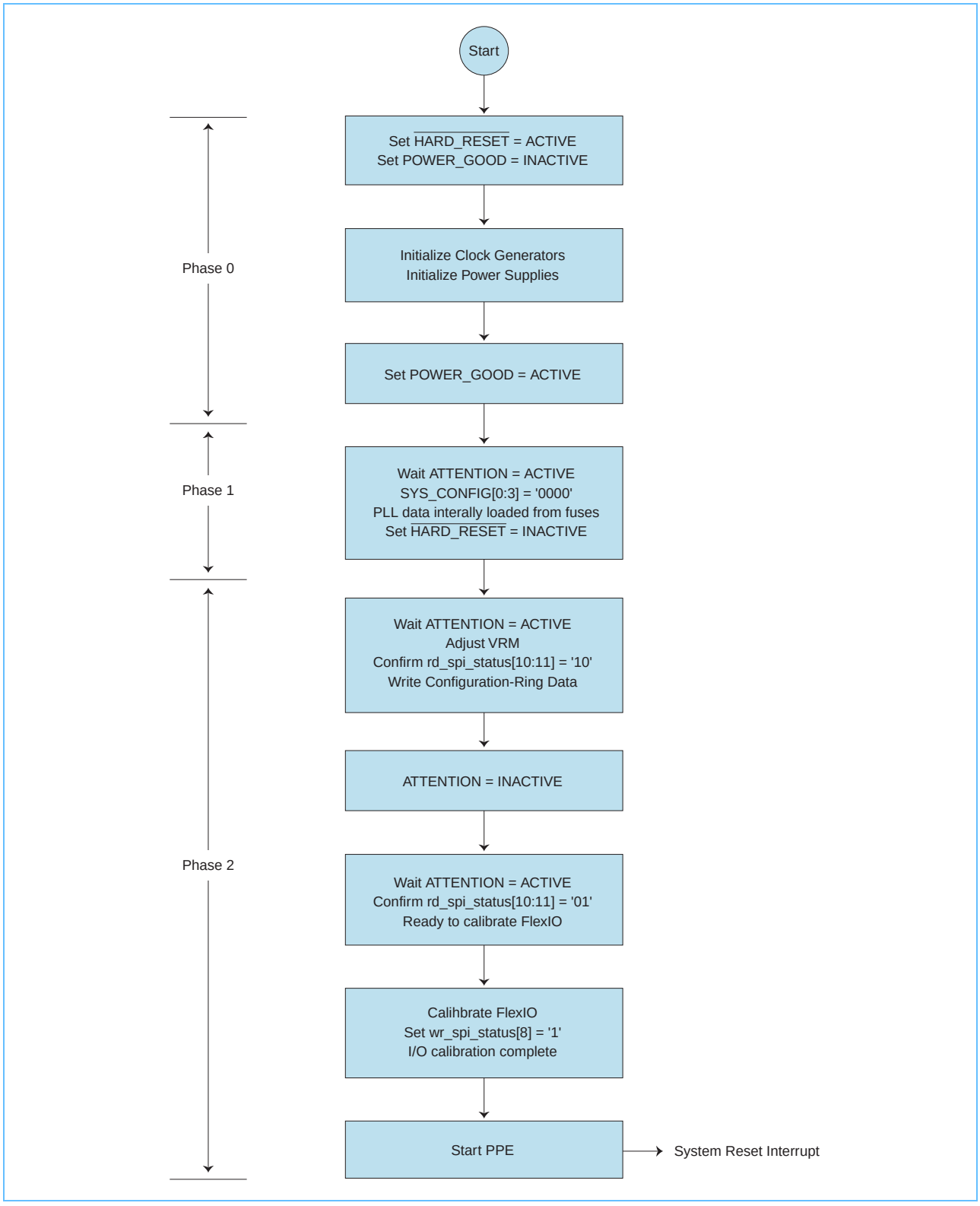


Table 2-1 summarizes the required steps in the POR sequence portion of the CBE initialization process.

Table 2-1. POR Sequence (Page 1 of 2)

POR Phase	CBE processor	I/O Device	System Controller
Phase 0			Drive POWER_GOOD inactive and HARD_RESET active. Activate reference clocks.
			Wait the minimum time (see *Cell Broadband Engine Datasheet*) after power supply is in regulation and reference clocks stable. Drive POWER_GOOD active.
	Detect start of POR.		
Phase 1	Scan initial state of MMIO and SPR registers to their POR values listed in the *Cell Broadband Engine Registers* document.		
	Read SYS_CONFIG[0:3] = '0000'. Write nominal PLL data from fuses into internal PLL configuration latches. Wait for HARD_RESET to become inactive.		
			Wait the minimum time (see *Cell Broadband Engine Datasheet*) after POWER_GOOD goes active.
			Drive HARD_RESET inactive.

Table 2-1. POR Sequence (Page 2 of 2)

POR Phase	CBE processor	I/O Device	System Controller
Phase 2	Continue initialization when HARD_RESET goes inactive		Wait for ATTENTION signal to become active.
	Activate ATTENTION to request configuration data.		
			Read SPI status register to determine reason for the ATTENTION signal. rd_spi_status[10:11] should be ‘10’.
			Read Voltage ID information from SPI. Adjust voltage as needed. Wait for power supply to stabilize.
			Write Configuration-Ring data through the wr_config_ring SPI register.
			Wait for ATTENTION signal.
	Continue internal initialization, which includes initializing the FlexIO and XIO PLLs.		
	Activate ATTENTION to indicate calibration is required.		
			Read SPI status register to determine reason for the ATTENTION signal. rd_spi_status[10:11] should be ‘01’.
	Participate in FlexIO calibration.	Participate in FlexIO calibration.	Calibrate FlexIO.
			Calibration complete. Notify the CBE processor by writing wr_spi_status[8] = ‘1’.
	Complete internal initialization		End of POR sequence.
	System reset interrupt (start PPE).		

2.1.2 Reset Detection

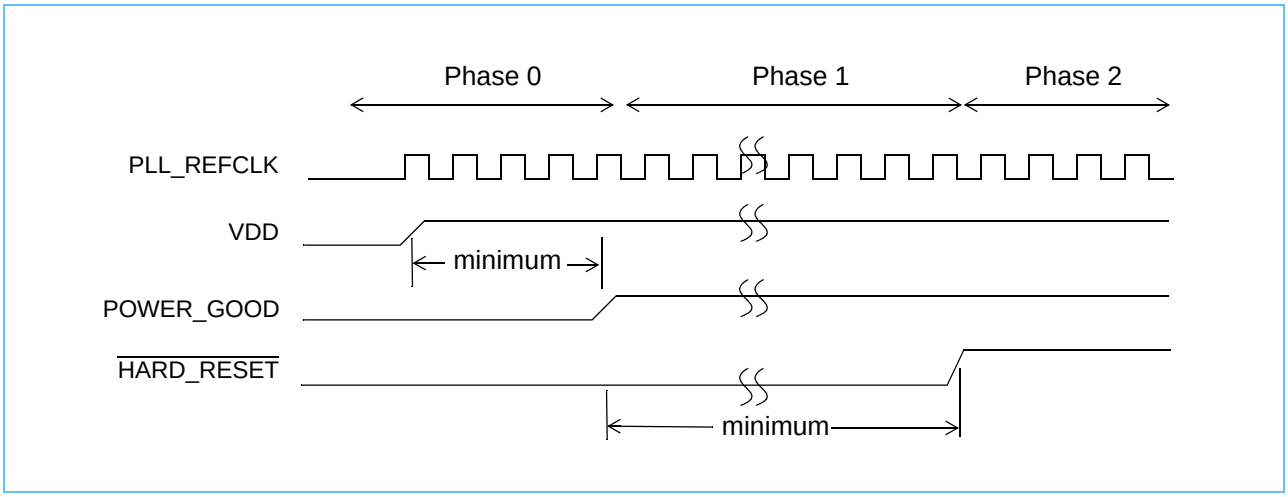
The beginning phase of initialization is referred to as Power-On Reset (POR). The initial application of power to the CBE processor, with the appropriate external signals active, starts the POR sequence. See the *Cell Broadband Engine Datasheet* for details on starting and stabilizing the power supplies and timing requirements on the POWER_GOOD and HARD_RESET signals.

There are two methods for detecting a reset condition:

- Cold start (cold reset).
- Warm start (warm reset).

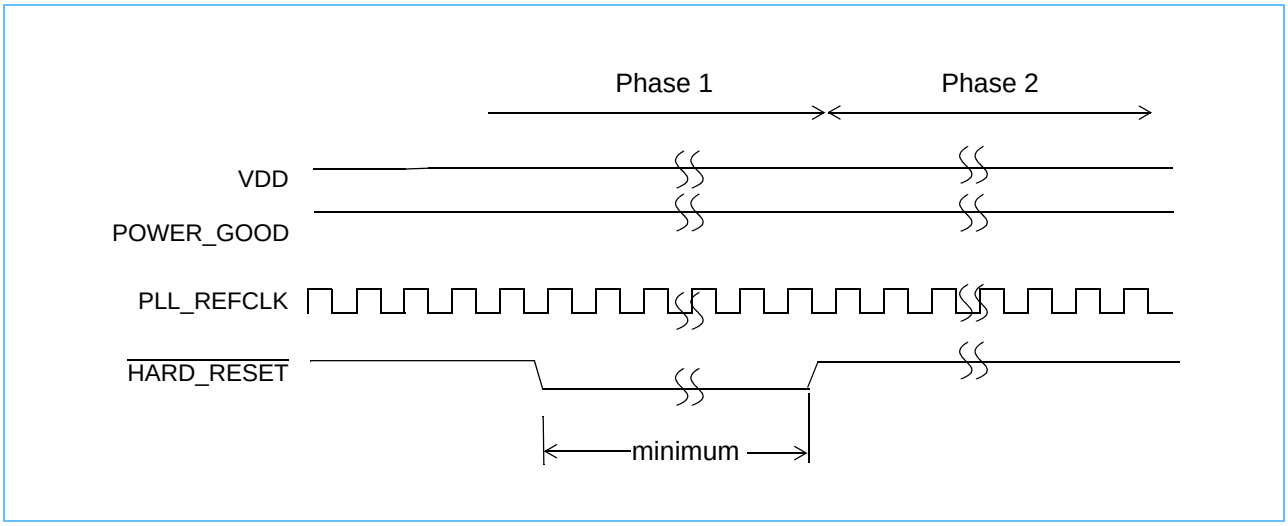
A cold start occurs when the CBE processor starts from a power-off state. A full POR sequence must be performed. The transition of the POWER_GOOD signal from inactive to active, with the HARD_RESET signal active, results in a cold start. *Figure 2-3* shows the timing. Refer to the *Cell Broadband Engine Datasheet* for minimum values.

Figure 2-3. Power-On Reset: POWER_GOOD Inactive-to-Active Transition



A warm start occurs when the CBE power supplies and reference clocks are all at a valid level, and the HARD_RESET signal transitions from inactive to active while the POWER_GOOD signal remains active. *Figure 2-4* shows the timing. The POR state machine inside the CBE processor treats the cold start and warm start methods identically.

Figure 2-4. Power-On Reset Detection: HARD_RESET Inactive-to-Active Transition



2.1.3 POR Phase 0

POR Phase 0 occurs only on a cold start. The events of POR Phase 0 are external to the CBE processor. Almost nothing occurs within the CBE processor itself.

To start POR Phase 0, the System Controller must drive the POWER_GOOD and HARD_RESET signals to zero. Next, the power supplies and reference clocks must be activated. The CBE power supplies must be turned on in the following order:

1. VDD_IO (I/O voltage supplies).

2. VDD (CBE core voltage supply).
3. VDD_A (Analog voltage supply).

At this point, Voltage ID (VID) information for the Voltage Regulation Module (VRM) is not available for readout. Therefore, at this point, the VRM needs to be set to a default VID value (see the *Cell Broadband Engine Datasheet*). Chip VID information becomes available later in the POR sequence.

A minimum time (see the *Cell Broadband Engine Datasheet*) after the power supplies and the reference clocks have stabilized, the POWER_GOOD signal can be raised to active.

2.1.4 POR Phase 1

The CBE processor uses four signals, SYS_CONFIG[0:3], to determine the internal steps that occur during the POR sequence. For normal booting of the CBE processor, these pins are tied to GND. These signals are tied to GND or MC2_VDDIO through resistors on the system board.

Phase 1 of the POR sequence begins with the transition of the POWER_GOOD signal from inactive to active. During phase 1, the PLL configuration register will be set up from internal storage (fuses) as a result of the SYS_CONFIG[0:3] pins being set to ’0000’. The PLL configuration register is an internal register that is not accessible in the MMIO register space.

After a minimum time (see the *Cell Broadband Engine Datasheet*) has elapsed, counting from the rising edge of POWER_GOOD, the $\overline{\text{HARD\_RESET}}$ signal can be switched to inactive. This triggers the next phase (Phase 2).

2.1.5 POR Phase 2

Phase 2 of the POR sequence starts when the $\overline{\text{HARD\_RESET}}$ signal transitions from active to inactive. The following steps occur during POR Phase 2:

1. Adjust VRM according to VID information (as described in *Section 2.1.5.1* on page 34) and load the Configuration Ring (as described in *Section 2.1.5.2* on page 34).
2. Calibrate the FlexIO interface, as described in *Section 2.1.5.3* on page 35.
3. Start the PPE (see *Section 2.2* on page 52).

The adjustment of the VRM according to VID information and the Configuration-Ring load are treated as one event notification from the CBE processor.

The CBE processor signals the System Controller that configuration data is required by activating the ATTENTION signal. The System Controller then reads bits [10:11] of the Read SPI Status Register (`rd_spi_status`) to determine the cause of the ATTENTION signal. The VRM value must be set to the VID value stored in CBE processor prior to loading the Configuration-Ring data.

The following sections describe the details of Steps 1 and 2, above, of POR Phase 2.

2.1.5.1 *VRM adjustment with VID information*

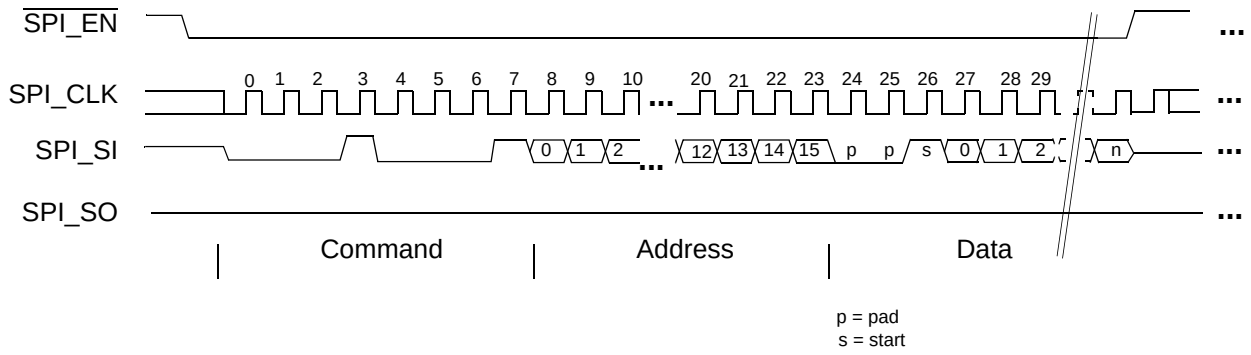
When the CBE processor activates the ATTENTION signal, the System Controller reads the SPI Read Voltage ID (`rd_VID`) register. This register contains 8 bits of VID data, to which the VRM should be set. At this point, the CBE voltage can be adjusted. After the CBE core VDD voltage has stabilized to the new VRM setting (stabilization is dependent on the VRM chosen), the System Controller proceeds with loading the Configuration Ring. See the *Cell Broadband Engine Datasheet* for valid VID settings and for calibration data for LTD (Linear Thermal Diode).

2.1.5.2 *Configuration-Ring Load*

The Configuration Ring provides chip-level configuration information, as described in *Section 4 Configuration Ring* on page 119. This ring is defined as a write-only location, the SPI Write Configuration Ring (`wr_config_ring`) register, at SPI register address x0001 in the CBE Pervasive Logic. The Configuration Ring can only be written by the System Controller during the POR sequence. The System Controller reads the Read SPI Status Register (`rd_spi_status`) to determine when the Configuration Ring can be written. If `rd_spi_status[10:11]` = '10', then the CBE processor is ready for the Configuration-Ring data to be loaded.

Figure 2-5 shows the timing of the Configuration-Ring load, which involves a read from the source and a write to SPI registers.

Figure 2-5. Configuration-Ring Load



The Configuration-Ring write operation differs from writes to other SPI registers in several ways:

- The length of the Data portion is flexible. This is not the case for all other SPI registers. The CBE processor determines the end of the Configuration-Ring data not by fixed length, but by a Start Bit plus the number of bits in the Configuration Ring, as explained below.
- Configuration-Ring data must be scanned in reverse bit order. The Configuration Ring is 2697 bits long (0:2696). Data is scanned in from bit 2696 (least-significant bit) to bit 0 (most-significant bit). Therefore, the Data portion in *Figure 2-5* will have a Start Bit followed by bit 2696, bit 2695, and so forth, to bit 0. This only applies to the Data portion of the SPI protocol, and not for the Command or Address portions.
- Access to the `wr_config_ring` SPI register is only effective one time during POR, when requested explicitly from the CBE processor. An attempt to write this register during any other time has no effect.

The Data portion of the Configuration-Ring write shown in *Figure 2-5* consists of the following four elements, in order:

- Padding Bits.
- Start Bit.
- Configuration-Ring Data.
- Trailing Bits.

Padding and trailing bits may be needed by a system controller to byte-align its Configuration-Ring data (or any other data). No bits on the Configuration Ring before the first '1' must be looked at, and similarly, after the length of the Configuration Ring has been shifted in from the first '1', any additional bits will be ignored.

Padding Bits are a string of ‘0’s. The string can be of any length, including 0. All padding bits are ignored. The Start Bit is the first ‘1’ that appears in the data portion. The CBE processor looks for the Start Bit at the end of the configuration chain while scanning the data though it’s internal latches; the CBE processor stops scanning when it finds the Start Bit. Trailing Bits (any additional bits that appear after the Configuration-Ring bits) can be of any length and are ignored. The minimum length of the data portion must be one Start Bit plus 2697 Configuration-Ring bits. There is no maximum number of bits—Padding Bits and Trailing Bits can be of any length.

2.1.5.3 *FlexIO Bit and Byte Calibration (I/O Training)*

In this document, the terms *calibration* and *training* are synonyms. The CBE FlexIO interfaces perform two types of calibration at POR:

- *Bit Calibration*—This adjusts the bits within each 8-bit-wide Rambus channel for differences in circuit, wiring, and loading delays between the multiple bits of the Rambus channel. Bit calibration also calibrates driver current and driver impedance, and it levels the eight data bits to center the data eye around the clock edges.
- *Byte Calibration*—This levels the groups of 8-bit channels that make up the FlexIO interfaces (IOIF0 and IOIF1). Byte calibration also establishes proper envelope framing on the interface by detecting the location of the start-of-envelope pattern.

Only the FlexIO interfaces are calibrated during the POR Sequence. The XIO memory interface is calibrated during the Firmware Sequence, as described in *Section 2.2.2.1* on page 58.

After the System Controller has finished writing the Configuration-Ring data, the System Controller again waits for the ATTENTION signal to go active on the CBE processor. When the ATTENTION signal is active, the System Controller reads the SPI Status (`rd_spi_status`) register (*Section 3.4.1.1* on page 107) to see if `rd_spi_status[10:11]` = ’01’, indicating that the CBE processor is ready to start FlexIO calibration (see *Section 2.3* on page 84 if `rd_spi_status[10:11]` is not ’01’). The System Controller then sends a series of SPI commands to the CBE processor to do the calibration. Bit-calibration is done first, followed by byte-calibration.

When the code shown in *FlexIO Bit Calibration Code* on page 40 and *FlexIO Byte Calibration Code* on page 46 completes, the System Controller sets `wr_spi_status[8]` = ‘1’ to indicate that I/O calibration is complete. The POR state machine then sets the status in the `rd_por_status[8:9]` to reflect this.

As mentioned in the *Preface* on page 11, the sample code shown in this document is an example only and may need to be changed, depending on a specific system configuration and chip revision used. See the *I/O Reference Documentation* on page 11 for more information.

Header File for Calibration Code

The following header file is used in the FlexIO calibration bit and byte code. The calibration code is written for a CBE processor to CBE processor calibration configuration on IOIF0. Code is not provided for a CBE processor to Support Chip calibration configuration, because to do requires an assumption regarding support-chip registers, and this document does not describe any specific support chips.

```
/*************************************************************************
© Copyright International Business Machines Corporation, Sony Computer Entertainment
Incorporated, Toshiba Corporation 2005
All Rights Reserved


 FILENAME    : spi_lib.h
 DESCRIPTION : Support library for SPI/MMIO register accesses


*************************************************************************/

#ifndef _SPI_LIB_H
#define _SPI_LIB_H

#include "types.h"

typedef unsigned int   uint32;
typedef unsigned char  uchar;
typedef unsigned short ushort;
typedef unsigned short uint16;
typedef unsigned char  uint8;
typedef int            int32;
typedef short          int16;

#define uint64   struct UINT64

/************************************************/
/* SPI Commands and Addresses                   */
/* Ref: Pervasive-functional, ver 0.91, Table 5-13 */
/* - extended to add XCG and VRM                */
/************************************************/
#define CHIP_ID0               0x00
#define CHIP_ID1               0x04
#define CHIP_ID2               0x08
#define CHIP_ID3               0x0C

#define CHIP_BE                0x10
```

```
#define CHIP_BE0                CHIP_BE | CHIP_ID0
#define CHIP_BE1                CHIP_BE | CHIP_ID1

#define SPI_READ                0x00
#define SPI_WRITE               0x01

#define BE_PERVASIVE            0x0000
#define BE_MIC                  0xA000
#define BEI_BIC0_NCLK           0xD000
#define BEI_BIC1_NCLK           0xD400
#define BEI_EIB                 0xD800
#define BEI_IOC                 0xDC00
#define BEI_BIC0                0xE000
#define BEI_BIC1                0xF000

/*************************************************/
/* CBE Pervasive Registers                       */
/*                                               */
/*************************************************/
#define BE_ICB_POLL             0x0002
#define BE_RD_ICB_DATA          0x0010

/*********************/
/* CBE BIC Registers */
/*********************/
#define BED_RRAC_REGCTL         0x620
#define BED_RRAC_REGRDDAT       0x628

#define BIC_IF0INIT             0xe200
#define BIC_IF1INIT             0xe300

/************************/
/* FlexIO Bus Registers */
/************************/
#define BED_Lnk0_TransBytTrngCntl    0xf600
#define BED_Lnk1_TransBytTrngCntl    0xf608
#define BED_RecBytTrngCntl_Lnk0      0xf610
#define BED_RecBytTrngCntl_Lnk0      0xf618

/*********************************/
/* Rambus FlexIO(RRAC) Registers */
/*********************************/
#define RR_BLK0                 0x000
#define RR_BLK1                 0x100
#define RR_BLK2                 0x200
#define RR_BLK3                 0x300
#define RR_BLK4                 0x400
#define RR_BLK5                 0x500
#define RR_BLK6                 0x600
#define RR_BLK7                 0x700
```

```
#define RR_BLK8                 0x800
#define RR_BLK9                 0x900
#define RR_BLK10                0xA00
#define RR_BLK11                0xB00
#define RR_BLK12                0xC00
#define RR_BLK13                0xD00
#define RR_ALL_RX               0xE00
#define RR_ALL_TX               0xF00
#define RR_TX0                  0x000
#define RR_TX1                  0x100
#define RR_TX2                  0x200
#define RR_TX3                  0x300
#define RR_BX0                  0x700
#define RR_BX1                  0x800
#define RR_RX0                  0x900
#define RR_RX1                  0xA00
#define RR_RX2                  0xB00
#define RR_RX3                  0xC00

#define RR_PIN0                 0x00
#define RR_PIN1                 0x10
#define RR_PIN2                 0x20
#define RR_PIN3                 0x30
#define RR_PIN4                 0x40
#define RR_PIN5                 0x50
#define RR_PIN6                 0x60
#define RR_PIN7                 0x70
#define RR_GLOBAL               0x80
#define RR_PIN_ALL              0xE0
#define RR_G2_0_3               0x90
#define RR_G2_4_7               0xA0
#define RR_G2_0_7               0xB0
#define RR_RSRV                 0xF0

/* TX */
#define TX_PRBS_CTL             0x0
#define TX_CAL_CONFIG           0x1
#define TX_EQ1                  0x2
#define TX_EQ2                  0x3
#define TX_EQ3                  0x4

/* TX GLOBAL */
#define TX_PLL_CONFIGA          0x80
#define TX_PLL_CONFIGB          0x81
#define TX_PLL_STATUS           0x82
#define TX_ODT_MAN              0x83
#define TX_ODT_STATUS           0x84
#define TX_CONFIG               0x85
#define TX_FLNGTH               0x86
#define TX_BCTL                 0x87
```

```
#define TXCLK_EQ1               0x88
#define TXCLK_EQ2               0x89
#define TXCLK_EQ3               0x8A
#define TX_CTL                  0x8C

/* RX */
#define RX_PHASE_ADJ            0x0
#define RX_TCTL                 0x1
#define RX_PHASE                0x2
#define RX_ALIGN                0x3

/* RX GLOBAL */
#define RX_PLL_CONFIGA          0x80
#define RX_PLL_CONFIGB          0x81
#define RX_PLL_STATUS           0x82
#define RX_ODT_MAN              0x83
#define RX_ODT_STATUS           0x84
#define RX_CONFIG               0x85
#define RX_BCTL                 0x86
#define RX_ZPD_CONFIG           0x87
#define RX_CTL                  0x8C
#define RX_STATUS               0x8E

/* RX GLOBAL (G2) */
#define RX_TCAL_RANGE           0x0
#define RX_TCAL_CONTROL         0x1
#define RX_TCAL_STATUS          0x2
#define RX_TCAL_PF              0x3
#define RX_OS_COUNT             0x4
#define RX_FLNGTH               0x6

/* BX */
#define RRAC_ID                 0x0
#define BX_MAN                  0x1
#define BX_STATE                0x2
#define BX_CONFIG               0x5
#define BX_CTL                  0xC
#define BX_STATUS               0xE

/* DEBUG */
#define TX_DBG_00               0xF0
#define TX_DBG_01               0xF1
#define TX_DBG_02               0xF2
#define TX_DBG_03               0xF3
#define RX_DBG_00               0xF0
#define RX_DBG_01               0xF1
#define RX_DBG_02               0xF2
#define RX_DBG_03               0xF3
#define RX_DBG_09               0xF4
```

```
int delay_us(int n);
int delay_ms(int n);

int spi_write64(uint8 cmd, uint16 address, uint32 data_hi, uint32 data_lo );

uint8  spi_read8(uint8 cmd, uint16 address);
uint16 spi_read16(uint8 cmd, uint16 address);
uint32 spi_read32(uint8 cmd, uint16 address);
uint64 spi_read64(uint8 cmd, uint16 address);

int spi_poll8(uint8 cmd, uint16 address,  uint8 mask,  uint8 test);
int spi_poll16(uint8 cmd, uint16 address, uint16 mask, uint16 test);
int spi_poll64(uint8 cmd, uint16 address, uint32 mask_hi, uint32 mask_lo, uint32
test_hi, uint32 test_lo);

int spi_write_rrac(uint8 cmd, uint16 address, uint16 data );
uint16 spi_read_rrac(uint8 cmd, uint16 address);
int spi_poll_rrac(uint8 cmd, uint16 address, uint16 mask, uint16 test);

#endif /* _SPI_LIB_H */
```

FlexIO Bit Calibration Code

The following script code is written for FlexIO bit calibration between two CBE processors (BE0 and BE1)[1]:

```
/*************************************************************************
© Copyright International Business Machines Corporation, Sony Computer Entertainment
Incorporated, Toshiba Corporation 2005
All Rights Reserved

/*************************************************************************

FILENAME    : be2be.c
DESCRIPTION : Bit Calibration for CBE processor<-->CBE processor interface
*************************************************************************/
#include "spi_lib.h"


int IOIF0_bit_training(void)
{
   int rc;
   int chip_bex, be_chip;

   for (be_chip = 0; be_chip < 2; be_chip++) {
```

1. The calibration code is written for a CBE processor to CBE processor calibration configuration on IOIF0. Code is not provided for a CBE processor to Support Chip calibration configuration, because to do requires an assumption regarding support-chip registers, and this document does not describe any specific support chips.

```
        chip_bex = (CHIP_BE | (be_chip << 2)) << 16;

        /* Set VDDCR = 1.15V (CBE Only) */
        spi_write_rrac( chip_bex, RR_ALL_TX | RR_RSRV | TX_DBG_01, 0x001e);
        spi_write_rrac( chip_bex, RR_ALL_RX | RR_RSRV | RX_DBG_01, 0x001e);

        /* PRBS Seed Value for each pin 0 on tx 0-3 channel in CBE */
        spi_write_rrac(chip_bex, RR_ALL_TX | RR_PIN0 | TX_PRBS_CTL, 0xaaa0);
        spi_write_rrac(chip_bex, RR_ALL_TX | RR_PIN1 | TX_PRBS_CTL, 0x1110);
        spi_write_rrac(chip_bex, RR_ALL_TX | RR_PIN2 | TX_PRBS_CTL, 0x2220);
        spi_write_rrac(chip_bex, RR_ALL_TX | RR_PIN3 | TX_PRBS_CTL, 0x3330);
        spi_write_rrac(chip_bex, RR_ALL_TX | RR_PIN4 | TX_PRBS_CTL, 0x4440);
        spi_write_rrac(chip_bex, RR_ALL_TX | RR_PIN5 | TX_PRBS_CTL, 0x5550);
        spi_write_rrac(chip_bex, RR_ALL_TX | RR_PIN6 | TX_PRBS_CTL, 0x6660);
        spi_write_rrac(chip_bex, RR_ALL_TX | RR_PIN7 | TX_PRBS_CTL, 0x7770);

        /* TX Equalization Adjustment 1,2 and 3 */
        spi_write_rrac(chip_bex, RR_ALL_TX | RR_PIN_ALL | TX_EQ1, 0x7f87);
        spi_write_rrac(chip_bex, RR_ALL_TX | RR_PIN_ALL | TX_EQ2, 0x3ae3);
        spi_write_rrac(chip_bex, RR_ALL_TX | RR_PIN_ALL | TX_EQ3, 0x01fe);

        /* Bits (1:0) = 00 = 2:1 Serialization */
        spi_write_rrac(chip_bex, RR_GLOBAL | RR_ALL_TX | TX_CONFIG, 0x0000);

        /* Frame Pattern Length  */
        spi_write_rrac(chip_bex, RR_GLOBAL | RR_ALL_TX | TX_FLNGTH, 0x0040);

        /* TX Bias Control (15:8) TX Pre-Driver Bias, (7:0) TX Driver Bias Control  */
        spi_write_rrac(chip_bex, RR_GLOBAL | RR_ALL_TX | TX_BCTL, 0x0808);

        /* TX Clk Equalization Adjustment 1 */
        spi_write_rrac(chip_bex, RR_GLOBAL | RR_ALL_TX | TXCLK_EQ1, 0x001C);

        /* TX Ctl Reg - Driver enable, ODT Enable, Pattern Xmit, IO On  */
        spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX0 | TX_CTL, 0x0039);
        spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX1 | TX_CTL, 0x0039);
        spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX2 | TX_CTL, 0x0039);
        spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX3 | TX_CTL, 0x0039);

        /* TX Pll Configuration A at default */
        spi_write_rrac(chip_bex, RR_GLOBAL | RR_ALL_TX | TX_PLL_CONFIGA, 0x0000);

        /* TX Pll Configuration B at default  0x0000 is full rate mode */
        spi_write_rrac(chip_bex, RR_GLOBAL | RR_ALL_TX | TX_PLL_CONFIGB, 0x0040);

        /* RX Pll Configuration B at default 0x0040 is for Full Rate Mode */
        spi_write_rrac(chip_bex, RR_GLOBAL | RR_ALL_RX | RX_PLL_CONFIGB, 0x0000);

    /* RX Configuration at default - 2:1 Serialization Mode */
```

```
spi_write_rrac(chip_bex, RR_GLOBAL | RR_ALL_RX | RX_CONFIG, 0x0000);

/* RX Receiver Gain Control */
spi_write_rrac(chip_bex, RR_GLOBAL | RR_ALL_RX | RX_BCTL, 0x00B4);

/* RX Ctl Reg - ODT enable, wait after phase cal, IO on  */
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX0 | RX_CTL, 0x0029);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX1 | RX_CTL, 0x0029);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX2 | RX_CTL, 0x0029);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX3 | RX_CTL, 0x0029);

/* RX Tcal Ctl - Single Step Enable, Single Step, byte count enable, 7 bit prbs
select, Level calibration enabled, parallel calibration enabled, phase
calibration enabled, all RX channels enabled  */
spi_write_rrac(chip_bex, RR_ALL_RX | RR_G2_0_7 | RX_TCAL_CONTROL, 0x00FE);

/* 128 parallel calibration compares, pass/fail of 40% UI */
spi_write_rrac(chip_bex, RR_ALL_RX | RR_G2_0_7 | RX_TCAL_PF, 0x8033);

/* 256 samples at each phase offset  */
spi_write_rrac(chip_bex, RR_ALL_RX | RR_G2_0_7 | RX_OS_COUNT, 0x0100);

/* Frame Pattern Length  */
spi_write_rrac(chip_bex, RR_ALL_RX | RR_G2_0_7 | RX_FLNGTH, 0x0040);

}

for (be_chip = 0; be_chip < 2; be_chip++) {

   chip_bex = (CHIP_BE | (be_chip << 2)) << 16;


   /* ODT and Current Calibration */
   spi_write_rrac(chip_bex, RR_BX0 | BX_CTL, 0x0005);
   spi_write_rrac(chip_bex, RR_BX1 | BX_CTL, 0x0005);

   /* Check for BX Block Ready */
   spi_poll_rrac(chip_bex, RR_BX0 | BX_STATUS, 0x0800, 0x0800);
   spi_poll_rrac(chip_bex, RR_BX1 | BX_STATUS, 0x0800, 0x0800);

   /* Deassert reset and BX current and ODT enable */
   spi_write_rrac(chip_bex, RR_BX0 | BX_CTL, 0x0003);
   spi_write_rrac(chip_bex, RR_BX1 | BX_CTL, 0x0003);

  /* Check RX calibration for complete and pass */
   spi_poll_rrac(chip_bex, RR_BX0 | BX_STATUS, 0x0003, 0x0003);
   spi_poll_rrac(chip_bex, RR_BX1 | BX_STATUS, 0x0003, 0x0003);


 /* Driver enable */
```

```
spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX0 | TX_CTL, 0x003b);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX1 | TX_CTL, 0x003b);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX2 | TX_CTL, 0x003b);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX3 | TX_CTL, 0x003b);

/* Check for PLL Lock */
spi_poll_rrac(chip_bex, RR_GLOBAL | RR_TX0 | TX_PLL_STATUS, 0x0001, 0x0001);
spi_poll_rrac(chip_bex, RR_GLOBAL | RR_TX1 | TX_PLL_STATUS, 0x0001, 0x0001);
spi_poll_rrac(chip_bex, RR_GLOBAL | RR_TX2 | TX_PLL_STATUS, 0x0001, 0x0001);
spi_poll_rrac(chip_bex, RR_GLOBAL | RR_TX3 | TX_PLL_STATUS, 0x0001, 0x0001);

/* Receiver enable */
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX0 | RX_CTL, 0x002b);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX1 | RX_CTL, 0x002b);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX2 | RX_CTL, 0x002b);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX3 | RX_CTL, 0x002b);

/* Check for PLL Lock */
spi_poll_rrac(chip_bex, RR_GLOBAL | RR_RX0 | RX_PLL_STATUS, 0x0001, 0x0001);
spi_poll_rrac(chip_bex, RR_GLOBAL | RR_RX1 | RX_PLL_STATUS, 0x0001, 0x0001);
spi_poll_rrac(chip_bex, RR_GLOBAL | RR_RX2 | RX_PLL_STATUS, 0x0001, 0x0001);
spi_poll_rrac(chip_bex, RR_GLOBAL | RR_RX3 | RX_PLL_STATUS, 0x0001, 0x0001);
}

for (be_chip = 0; be_chip < 2; be_chip++) {

   chip_bex = (CHIP_BE | (be_chip << 2)) << 16;

/* TX Digital Reset */
spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX0 | TX_CONFIG, 0x0020);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX0 | TX_CONFIG, 0x0000);

spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX1 | TX_CONFIG, 0x0020);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX1 | TX_CONFIG, 0x0000);

spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX2 | TX_CONFIG, 0x0020);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX2 | TX_CONFIG, 0x0000);

spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX3 | TX_CONFIG, 0x0020);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_TX3 | TX_CONFIG, 0x0000);

/* RX Digital Reset */
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX0 | RX_CONFIG, 0x0020);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX0 | RX_CONFIG, 0x0000);

spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX1 | RX_CONFIG, 0x0020);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX1 | RX_CONFIG, 0x0000);

spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX2 | RX_CONFIG, 0x0020);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX2 | RX_CONFIG, 0x0000);
```

```
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX3 | RX_CONFIG, 0x0020);
spi_write_rrac(chip_bex, RR_GLOBAL | RR_RX3 | RX_CONFIG, 0x0000);
}

/* RX Ctl Reg – ODT enable, wait after phase cal, RX Block Enable, IO on */
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_RX0 | RX_CTL, 0x002f);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_RX1 | RX_CTL, 0x002f);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_RX2 | RX_CTL, 0x002f);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_RX3 | RX_CTL, 0x002f);

spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_RX0 | RX_CTL, 0x002f);
spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_ RX1 | RX_CTL, 0x002f);
spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_ RX2 | RX_CTL, 0x002f);
spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_ RX3 | RX_CTL, 0x002f);

/* Check for Phase calibration complete, Phase calibration passed */

spi_poll_rrac(CHIP_BE0, RR_GLOBAL | RR_RX0 | RX_STATUS, 0x0003, 0x0003);
spi_poll_rrac(CHIP_BE0, RR_GLOBAL | RR_RX1 | RX_STATUS, 0x0003, 0x0003);
spi_poll_rrac(CHIP_BE0, RR_GLOBAL | RR_RX2 | RX_STATUS, 0x0003, 0x0003);
spi_poll_rrac(CHIP_BE0, RR_GLOBAL | RR_RX3 | RX_STATUS, 0x0003, 0x0003);

spi_poll_rrac(CHIP_BE1, RR_GLOBAL | RR_RX0 | RX_STATUS, 0x0003, 0x0003);
spi_poll_rrac(CHIP_BE1, RR_GLOBAL | RR_RX1 | RX_STATUS, 0x0003, 0x0003);
spi_poll_rrac(CHIP_BE1, RR_GLOBAL | RR_RX2 | RX_STATUS, 0x0003, 0x0003);
spi_poll_rrac(CHIP_BE1, RR_GLOBAL | RR_RX3 | RX_STATUS, 0x0003, 0x0003);



for (be_chip = 0; be_chip < 2; be_chip++) {

   chip_bex = (CHIP_BE | (be_chip << 2)) << 16;

   /* Manual Skip Select On */
   spi_write_rrac(chip_bex, RR_RX0 | RR_PIN_ALL | RX_TCTL, 0x0002);

   /* Manual Skip Select Off */
   spi_write_rrac(chip_bex, RR_RX0 | RR_PIN_ALL | RX_TCTL, 0x0000);

   /* Manual Skip Select On */
   spi_write_rrac(chip_bex, RR_RX1 | RR_PIN_ALL | RX_TCTL, 0x0002);

   /* Manual Skip Select Off */
   spi_write_rrac(chip_bex, RR_RX1 | RR_PIN_ALL | RX_TCTL, 0x0000);

   /* Manual Skip Select On  */
   spi_write_rrac(chip_bex, RR_RX2 | RR_PIN_ALL | RX_TCTL, 0x0002);

   /* Manual Skip Select Off  */
```

```
    spi_write_rrac(chip_bex, RR_RX2 | RR_PIN_ALL | RX_TCTL, 0x0000);

    /* Manual Skip Select On  */
    spi_write_rrac(chip_bex, RR_RX3 | RR_PIN_ALL | RX_TCTL, 0x0002);

    /* Manual Skip Select Off  */
    spi_write_rrac(chip_bex, RR_RX3 | RR_PIN_ALL | RX_TCTL, 0x0000);
}

/* Driver enable, ODT Enable, Calibration Framing Pattern Xmit, enable TX BClk,
IO On */
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_TX0 | TX_CTL, 0x003f);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_TX1 | TX_CTL, 0x003f);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_TX2 | TX_CTL, 0x003f);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_TX3 | TX_CTL, 0x003f);

spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_TX0 | TX_CTL, 0x003f);
spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_TX1 | TX_CTL, 0x003f);
spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_TX2 | TX_CTL, 0x003f);
spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_TX3 | TX_CTL, 0x003f);

/* ODT enable, proceed with parallel cal, RX Block Enable, IO on */
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_RX0 | RX_CTL, 0x0027);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_RX1 | RX_CTL, 0x0027);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_RX2 | RX_CTL, 0x0027);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_RX3 | RX_CTL, 0x0027);

spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_RX0 | RX_CTL, 0x0027);
spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_RX1 | RX_CTL, 0x0027);
spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_RX2 | RX_CTL, 0x0027);
spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_RX3 | RX_CTL, 0x0027);

/* check parallel calibration and slice levelization complete and passed  */

spi_poll_rrac(CHIP_BE0, RR_GLOBAL | RR_RX0 | RX_STATUS, 0x003f, 0x003f);
spi_poll_rrac(CHIP_BE0, RR_GLOBAL | RR_RX1 | RX_STATUS, 0x003f, 0x003f);
spi_poll_rrac(CHIP_BE0, RR_GLOBAL | RR_RX2 | RX_STATUS, 0x003f, 0x003f);
spi_poll_rrac(CHIP_BE0, RR_GLOBAL | RR_RX3 | RX_STATUS, 0x003f, 0x003f);

spi_poll_rrac(CHIP_BE1, RR_GLOBAL | RR_RX0 | RX_STATUS, 0x003f, 0x003f);
spi_poll_rrac(CHIP_BE1, RR_GLOBAL | RR_RX1 | RX_STATUS, 0x003f, 0x003f);
spi_poll_rrac(CHIP_BE1, RR_GLOBAL | RR_RX2 | RX_STATUS, 0x003f, 0x003f);
spi_poll_rrac(CHIP_BE1, RR_GLOBAL | RR_RX3 | RX_STATUS, 0x003f, 0x003f);

/* Driver enable, ODT Enable, enable core data, enable TX BClk, IO On */
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_TX0 | TX_CTL, 0x0037);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_TX1 | TX_CTL, 0x0037);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_TX2 | TX_CTL, 0x0037);
spi_write_rrac(CHIP_BE0, RR_GLOBAL | RR_TX3 | TX_CTL, 0x0037);
```

```
    spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_TX0 | TX_CTL, 0x0037);
    spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_TX1 | TX_CTL, 0x0037);
    spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_TX2 | TX_CTL, 0x0037);
    spi_write_rrac(CHIP_BE1, RR_GLOBAL | RR_TX3 | TX_CTL, 0x0037);
    }
```

FlexIO Byte Calibration Code

The following C-like code is an example of FlexIO byte calibration between two CBE processors (BE0 and BE1)[1]:

```
/*************************************************************************
© Copyright International Business Machines Corporation, Sony Computer Entertainment
Incorporated, Toshiba Corporation 2005
All Rights Reserved


FILENAME    : rrac_be_byte_training.c
DESCRIPTION : Byte Calibration for CBE processor<-->CBE processor interface

*************************************************************************/
#include "spi_lib.h"


int IOIF0_byte_training(void)
{
   int rc=0;

   /* Enable CBE RX byte training */
   spi_write64(CHIP_BE0, BED_RecBytTrngCntl_Lnk0, 0x86000000, 0x00000000);
   spi_write64(CHIP_BE1, BED_RecBytTrngCntl_Lnk0, 0x86000000, 0x00000000);

   /* Transmit latency set to 2.  Enable CBE TX byte trainig */
   spi_write64(CHIP_BE0, BED_Lnk0_TransBytTrngCntl, 0xc2000000, 0x00000000);
   spi_write64(CHIP_BE1, BED_Lnk0_TransBytTrngCntl, 0xc2000000, 0x00000000);

    rc = spi_poll64(CHIP_BE0, BED_RecBytTrngCntl_Lnk0, 0xff000000, 0x00000000,
0xe6000000, 0x00000000);
    rc += spi_poll64(CHIP_BE1, BED_RecBytTrngCntl_Lnk0, 0xff000000, 0x00000000,
0xe6000000, 0x00000000);

   /* turn off Byte Training Pattern XMIT_EN */
   spi_write64(CHIP_BE0, BED_Lnk0_TransBytTrngCntl, 0x82000000, 0x00000000);
   spi_write64(CHIP_BE1, BED_Lnk0_TransBytTrngCntl, 0x82000000, 0x00000000);

   /* turn off Byte Training Pattern RCV_EN */
```

1. The calibration code is written for a CBE processor to CBE processor calibration configuration on IOIF0. Code is not provided for a CBE processor to Support Chip calibration configuration, because to do requires an assumption regarding support-chip registers, and this document does not describe any specific support chips.

```
   spi_write64(CHIP_BE0, BED_RecBytTrngCntl_Lnk0, 0x66000000, 0x00000000);
   spi_write64(CHIP_BE1, BED_RecBytTrngCntl_Lnk0, 0x66000000, 0x00000000);

   /* Enable Reception/Transmission Layer */
   /* These next few command will start the CBE code execution */

   /* Enable Link Layer */
   spi_write64(CHIP_BE0, BIC_IFOINIT, 0x80000000, 0x00000000);
   spi_write64(CHIP_BE1, BIC_IFOINIT, 0x80000000, 0x00000000);

   /* enable Transport_Layer_Transmission_Enable */
   spi_write64(CHIP_BE0, BIC_IFOINIT, 0xc0000000, 0x00000000);
   spi_write64(CHIP_BE1, BIC_IFOINIT, 0xc0000000, 0x00000000);

   return rc;
}
```

Data-Link Controls

Data-link control initialization is not part of this initialization sequence. CBE firmware sets the expiration value for the envelop retry counter at 0x'0108', as specified in bits [0:15] of the `IFOTHR` register for the 4-byte BE-to-BE link using IOIF0.

The default values of the link timers are set at their maximum rate, with CRC and retry thresholds disabled. It should be possible to bring up any link with the default configurations, but at some point these values should be optimized for the configuration and platform requirements, which may vary between applications. Several registers need to be updated with data specific to the platform after the link is operational, and it is expected that the retry timers would be set at the same time. Also, if in secure mode, the link retry timers cannot be accessed through SPI.

Library File for Calibration Code

Subroutines from the following library file are used in the preceding calibration code.

```
/*************************************************************************
© Copyright International Business Machines Corporation, Sony Computer Entertainment
Incorporated, Toshiba Corporation 2005
All Rights Reserved


 FILENAME    : spi_lib.c
 DESCRIPTION : Support library for SPI register accesses.


*************************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "spi_lib.h"
```

```
#define NUM_POLL              10       /* number of attempts before polling fails */


/* bit_reverse16: reverse bit order for the given 16-bit value */
uint16 bit_reverse16(uint16 n) {

n = ((n & 0xff00) >> 8) | ((n & 0x00ff) << 8);/* flip bytes  */
n = ((n & 0xf0f0) >> 4) | ((n & 0x0f0f) << 4);/* flip nibbles   */
n = ((n & 0xcccc) >> 2) | ((n & 0x3333) << 2);/* flip bit pairs */
n = ((n & 0xaaaa) >> 1) | ((n & 0x5555) << 1);/* flip bits  */
return n;
}

/* delay_us: Time delay microseconds -- used to wait for hardware to stabilize */
int delay_us(int n)
{
   /*************************************************/
   /* system dependent routine to delay n microseconds */
   /*************************************************/
   return 0;
}

/* delay_ms: Time delay milliseconds -- used to wait for hardware to stabilize */
int delay_ms(int n)
{
   delay_us(n*1000);
   return 0;
}

/* spi_start: assert spi enable to start an external configuration bus command */
void spi_start(void)
{
   /***************************************/
   /* Drive SPI Enable (active low) to 0[1]  */
   /***************************************/
}

/* spi_stop: deassert spi enable to end an external configuration bus command */
void spi_stop(void)
{
   /***************************************/
   /* Drive SPI Enable (active low) to 1[2] */
   /***************************************/
}
```

1. This empty function should be supplied by the user. It is included here for completeness and to correlate with *Section 3 Serial Peripheral Interface* on page 91.
2. This empty function should be supplied by the user. It is included here for completeness and to correlate with *Section 3 Serial Peripheral Interface* on page 91.

```
/* spi_serial_write: format a stream of serial data for cmd/addr/data */
void spi_serial_write(uint32 data, int n)
{
   /*****************************************************************/
   /* Format data into serial stream for external configuration bus[1] */
   /*****************************************************************/
}

/* spi_serial_read: capture n bits of serial data    */
/*                  data is returned right justified */
uint32 spi_serial_read(int n)
{
   uint32 rtndata;

   /*****************************************/
   /* Capture n bits of returned serial data  */
   /*****************************************/
   return rtndata;
}


/* spi_write64: write data to specified address via the SPI bus */
int spi_write64(uint8 cmd, uint16 address, uint32 data_hi, uint32 data_lo)
{
   spi_start();
   spi_serial_write((cmd | SPI_WRITE) << 16 | address, 24);
   spi_serial_write(data_hi, 32);
   spi_serial_write(data_lo, 32);
   spi_stop();
   return 0;
}

/* spi_read8: read specified address via the SPI bus */
uint8 spi_read8(uint8 cmd, uint16 address)
{
   uint8 rtndata;

   spi_start();
   spi_serial_write(cmd << 16 | address, 24);
   rtndata = spi_serial_read(8) & 0x000000FF;
   spi_stop();
   return rtndata;
}

/* spi_read16: read specified address via the SPI bus */
uint16 spi_read16(uint8 cmd, uint16  address)
{
   uint16 rtndata;
```

1. This empty function should be supplied by the user. It is included here for completeness and to correlate with *Section 3 Serial Peripheral Interface* on page 91.

```
   spi_start();
   spi_serial_write(cmd << 16 | address, 24);
   rtndata = spi_serial_read(16) & 0x0000FFFF;
   spi_stop();
  return rtndata;
}

/* spi_read32: read specified address via the SPI bus */
uint32 spi_read32(uint8 cmd, uint16  address)
{
   uint32 rtndata;

   spi_start();
   spi_serial_write(cmd << 16 | address, 24);
   rtndata = spi_serial_read(32);
   spi_stop();
   return rtndata;
}

/* spi_read64: read specified address via the SPI bus */
uint64 spi_read64(uint8 cmd, uint16 address)
{
  uint64 rtndata;

   spi_start();
   spi_serial_write(cmd << 16 | address, 24);
   rtndata.msb = spi_serial_read(32);
   rtndata.lsb = spi_serial_read(32);
   spi_stop();
  return rtndata;
}

/* spi_poll8: poll specified address until bit(s) specified by mask are set */
int spi_poll8(uint8 cmd, uint16 address, uint8 mask, uint8 test) {
   uint8 data;
   int n = NUM_POLL;

   data = spi_read8(cmd, address);
   while ((data & mask) == (mask & test)) {
      delay_us(1000);
      data = spi_read8(cmd, address);
      if (--n == 0) {
         printf("spi_poll8: Poll failed.  cmd=%02x, address=%04x, mask=%02x,
            test=%02x\n", cmd, address, mask, test);
         exit(1);
      }
   }
   return 0;
}
```

```
/* spi_poll16: poll specified address until bit(s) specified by mask are set */
int spi_poll16(uint8 cmd, uint16 address, uint16 mask,  uint16 test) {
   uint16 data;
   int n = NUM_POLL;

   data = spi_read16(cmd, address);
   while ((data & mask) == (mask & test)) {
      delay_us(1000);
      data = spi_read16(cmd, address);
      if (--n == 0) {
         printf("spi_poll16: Poll failed.  cmd=%02x, address=%04x, mask=%04x,
            test=%04x\n", cmd, address, mask, test);
         exit(1);
      }
   }
   return 0;
}

/* spi_poll64: poll specified address until bit(s) specified by mask are set */
int spi_poll64(uint8 cmd, uint16 address, uint32 mask_hi, uint32 mask_lo,
       uint32 test_hi, uint32 test_lo) {
   uint64 data;
   int n = NUM_POLL;

   data = spi_read64(cmd, address);
   while ((data.msb & mask_hi != mask_hi & test_hi) | (data.lsb & mask_lo != mask_lo &
         test_lo)) {
      delay_us(1000);
      data = spi_read64(cmd, address);
      if (--n == 0) {
         printf("spi_poll64: Poll failed.  cmd=%02x, address=%04x, mask=%08x_%08x,
            test=%08x_%08x\n", cmd, address, mask_hi, mask_lo, test_hi, test_lo);
         exit(1);
      }
   }
   return 0;
}

/* spi_write_rrac: write data to specified FlexIO (RRAC) address via the SPI bus */
int spi_write_rrac(uint8 cmd, uint16 address, uint16 data)
{
   uint16 addr_reverse;

   addr_reverse = bit_reverse16(address & 0x00000FFF) >> 4;
   spi_write64(BEI_BIC1 | cmd, BED_RRAC_REGCTL, 0x40000000 | addr_reverse << 16 |
       bit_reverse16(data), 0);
   return 0;
}
```

```
/* spi_read_rrac: read data from specified FlexIO (RRAC) address */
uint16 spi_read_rrac(uint8 cmd, uint16 address) {
   uint32 rdData;
   uint16 rtnData, addr_reverse;

   addr_reverse = bit_reverse16(address & 0x00000FFF) >> 4;

   /* write requested FlexIO (RRAC) address to BIC RRAC interface control register */
   spi_write64(cmd | BEI_BIC1, BED_RRAC_REGCTL, addr_reverse << 16, 0);

   /* dummy read to send read request to BIC */
   spi_read64(cmd | BEI_BIC1, BED_RRAC_REGRDDAT);

   /* wait for indirect access to complete */
   spi_poll8(cmd | BE_PERVASIVE, BE_ICB_POLL, 0x80, 0x80);

   /* read data from BIC FlexIO (RRAC) data register */
   rdData = spi_read32(cmd | BE_PERVASIVE, BE_RD_ICB_DATA);
   rtnData = bit_reverse16(rdData >> 16);

   return rtnData;
}

/* spi_poll_rrac: poll register in FlexIO (RRAC) until bit(s) specified by mask are
set */
int spi_poll_rrac(uint8 cmd, uint16 address, uint16 mask, uint16 test) {
   uint16 data;
   int n = NUM_POLL;

   data = spi_read_rrac(cmd, address);
   while (data & mask != mask & test) {
      delay_us(1000);
      data = spi_read_rrac(cmd, address);
      if (--n == 0) {
         printf("spi_poll_rrac: Poll failed.  cmd=%02x, address=%04x, mask=%08x,
             test=%08x\n", cmd, address, mask, test);
         exit(1);
      }
   }
   return 0;
}
```

2.2 Firmware Sequence

After the System Controller has notified the CBE processor that I/O calibration has completed (as indicated by `wr_spi_status[8]` = ‘1’), the POR state machine sets `rd_por_status[8]` to ‘0’ and `rd_por_status[9]` to ‘1’ to indicate that I/O calibration is not active and has completed. The POR

state machine is then finished with the POR Sequence and instructs the PPE to begin executing code, which is the beginning of the Firmware Sequence. A flowchart and pseudocode for the sequence are given in *Section 2.2.1*.

Because HID1 SPR defaults to all zeros during POR, the PPE takes a System Reset Interrupt and starts thread 0 from the address specified in the *PPE SReset Vector* field of the Configuration Ring. As a result of the System Reset Interrupt, the Hypervisor and 64-Bit-Mode bits, MSR[HV] and MSR[SF], are both set to '1', so that the PPE comes up in hypervisor mode.

From this point forward, the System Controller does not participate in CBE Initialization. If the ATTENTION signal transitions to active after the POR sequence is complete, it indicates that an error condition has occurred, for which the CBE processor needs the System Controller's help. In this case, the System Controller must read the rd_spi_status register to determine what caused the ATTENTION signal and take appropriate action.

2.2.1 Firmware-Sequence Flowchart and Pseudocode

Figure 2-6 on page 55 shows a flowchart for the Firmware Sequence. The associated code, which follows this figure, would be written in PPE assembler code, but is shown here as pseudo code for readability and because some external devices, such as an I/O bridge chip and ROM, would exist in a system but are beyond the scope of this document. The XIO and MIC initialization is part of the PPE Firmware Sequence and is discussed in more detail in the next sections.

The flowchart and pseudocode assume that the system has already initialized an interface to an I/O bridge chip by means of the SPI interface, and that the following firmware is already loaded into a ROM attached to the I/O bridge chip, such that the entry point is at the address specified in the *PPE SReset Vector* field of the Configuration Ring, with the low-order address bits equal to x'100'.

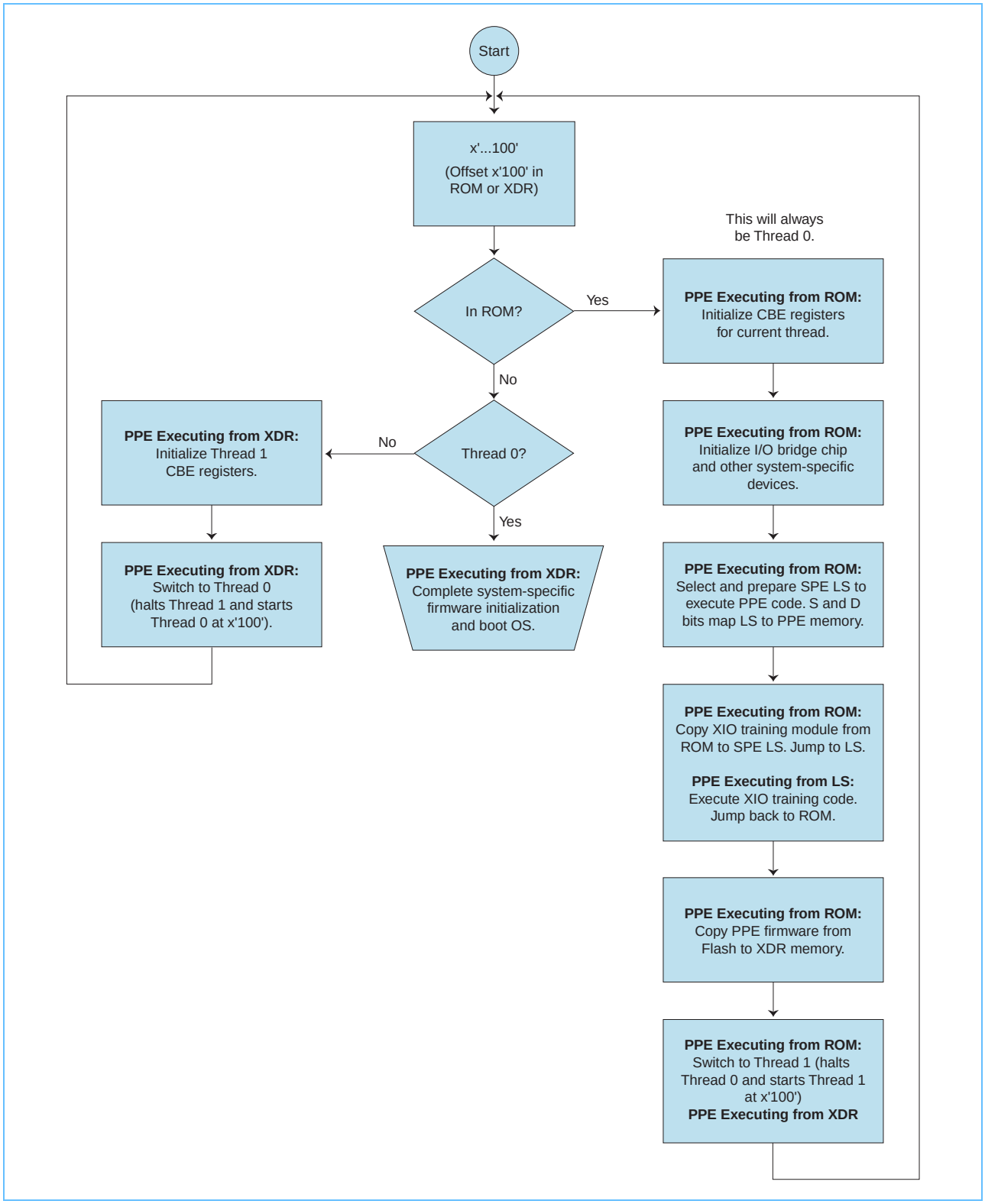
The firmware is executed from ROM the first time through the sequence. Then, the HID1 register is initialized so that thread 0 will go to address x'0000000100' the next time thread 0 is used (which, in this example, is in the XDR memory space). After calibrating the XIO interface and initializing the XDR DRAM, using the Local Store (LS) memory space for one of the SPEs as memory for the PPE, the PPE copies the code from ROM to XDR memory and then starts thread 1.

The calibration of the XIO interface and initializing of XDR DRAM uses the Local Store (LS) memory space of one SPE as instruction and data memory for the PPE. The XIO calibration is done from an SPE's LS rather than ROM, because using this read/write memory simplifies creation of a stack in C code. To use an SPE's LS, the PPE first enables LS-address translation by setting the S and D bits in the MFC_SR1 register for the selected SPE (see the Memory Map chapter of the *Cell Broadband Engine Programming Handbook* for details). As an alternative, XIO calibration could be done from ROM without using an SPE's LS memory, but in that case the code would probably be written in assembly language.

After calibrating the XIO interface and initializing memory with good ECC, the PPE copies the code from ROM to the PPE's XDR memory space and then starts thread 1 to initialize registers for thread 1. Thread 1 always starts at address x'0000000100', which will be in XDR memory. Thread 1 initializes registers used by thread 1, and then the code switches to thread 0 again (also at x'0000000100') to complete the rest of the system firmware initialization. That ends the Firmware Sequence. The next step would be to load the operating system, which is beyond the scope of this document.

The sample firmware uses the LS for an SPE to hold XIO training code, and this documentation describes that implementation. However, there is no technical reason for not using PPE cache to hold XIO training code.

Figure 2-6. PPE Firmware Flowchart



The Firmware Sequence pseudocode is:

```
entry: (This is at offset 0x100 and could be in ROM or XDR)

IF executing in ROM
  Call init_regs
  Initialize I/O bridge chip and other system-specific devices
  Select SPE
  Call init_spe
  Copy PPE XIO/XDR initialization code into SPE local store
  Call XIO/XDR initialization function (code located in SPE)
  Clear the NCRSO bit in MMIO EIB_Cfg register
  Copy ROM code into XDR
  Stop thread 0 and start thread 1 (thread 1 starts at 0x100 in XDR memory)
ELSE
  IF thread 1 executing
    Call init_regs
    Stop thread 1 and start thread 0 (thread 0 starts at 0x100 in XDR memory)
  ELSE
    Complete initialization for rest of system components
    Load and execute OS
  ENDIF
ENDIF
```

The function `init_regs` initializes CBE registers required for proper execution of the low-level firmware. Caution must be used when initializing the HID registers, because they are not duplicated for each thread. In other words, the HID registers should only be initialized by the first running thread (in this case, thread 0). Other registers that are duplicated for each thread must be initialized by code running within the context of their respective thread. See the *Cell Broadband Engine Registers* document for information on whether or not a particular register is duplicated for multithreading.

```
init_regs:

// Set the syserr_wakeup, en_prec_mchhk, qattn_mode, en_syserr, and en_attn
// bits in HID0.
Write SPR HID0 with 0x0000_00AB_0000_0000

// Set the dis_sysrst_reg bit in HID1 to disable the thread 0 PPE SReset Vector //
address in the Configuration Ring, which will make 0x100 the reset vector for //
thread 0. This will also enable the L1 instruction cache.
Write SPR HID1 with 0x9C30_1040_0000_0000

// Enable the L1 Data cache.
Write SPR HID4 with the 0x0000_3F00_0000_0000 or'ed into the original value

// Set RMSC to 1110b which will set the real address boundary at 2TB
// (0x200_0000_0000_0000_0000).
```

```
Write SPR HID6 with the masking of the original value with 0xFFFF_FFC3_0000_0000 and
or'ing in 0x0000_0038_0000_0000

// Set the RMI bit in LPCR which makes the memory above the real address
// boundary (the RMSC in HID6) non-cacheable for data and guarded
Write SPR LPCR with 0x0000_0000_0000_0002 or'ed into the original value

// Set DISP_CNT to 4 and bits PBUMP, FPCF, and PSCTP
Write TSCR with 0x200D_0000

// Set TTIM to 0x01F4
Write SPR TTR with 0x0000_0000_0000_01F4

// Disable the Timebase by clearing tb_enable in HID6
Write HID6 with the original value and'ed with 0xFFFE_FFFF_FFFF_FFFF

// Write TBR with the Timebase_mode (internal or external time base sync mode)
// and the Timebase_setting which is the divisor for the internal time base if
// in internal sync mode.
Write MMIO TBR with the Timebase_setting and Timebase_mode

// Set tb_enable in HID6 to enable the time base
Write HID6 with the original value or'ed with 0x0001_0000_0000_0000

return
```

The function `init_spe` initializes an SPE so that PPE code can be copied into it's local store (LS) and executed by the PPE. This is done by choosing an SPE and setting the S and D bits in the MFC_SR1 register (see the *Cell Broadband Engine Architecture* document for bit definitions) to allow the LS to be memory-mapped into PPE space. This allows the direct copy into, and execution out of, LS by the PPE.

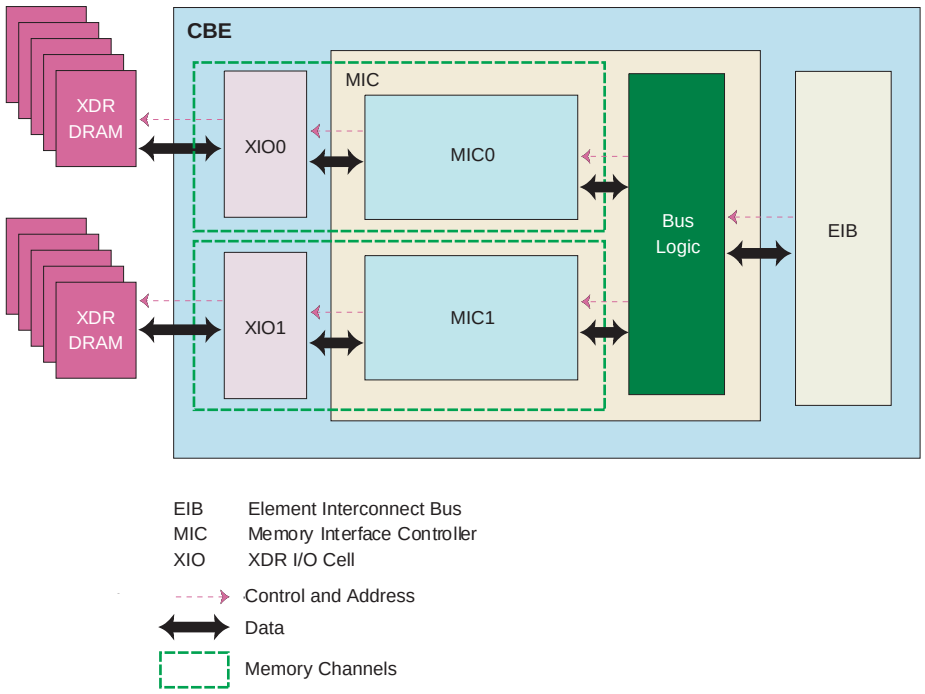
```
init_spe:

// Write the SPE's MFC_SR1 register with S and D bits set.
Write MMIO MFC_SR1 with 0x000_0000_0000_0021

// Set NCRS0 and NCRS1 bits in EIB_Cfg to make noncoherent ranges 0 and 1
// global
Write MMIO EIB_Cfg with 0x0018_0000_0000_0000 or'ed with the original value
return
```

2.2.2 Initialization of MIC, XDR I/O Cells, and XDR DRAM

This section provides information and an example of the initialization of the Memory Interface Controller (MIC), the XDR IO Cell (XIO). It also describes the initialization of the external XDR DRAM chips. The MIC is the interface from the Element Interconnect Bus (EIB) and the XDR

memory channels. The XIO interfaces between the MIC and the external XDR DRAM chips. *Figure 2-7* on page 58 shows the major pieces of this subsystem. The two memory channels, 0 and 1, are independently operated and join together inside bus logic in the MIC.

Figure 2-7. Memory Subsystem



The MIC initialization can be done from either PPE code or SPE code. However, the pseudo code in *Section 2.2.1* on page 53 assumes that the MIC initialization code is executed by the PPE from an SPE's LS memory space[1]. The example also assumes one CBE processor, in the configuration outlined in *Section 1.2 Clock Domains* on page 21.

For details about XDR initialization, see the *Rambus XDR Initialization Guide (DL-0178)* supplied by Rambus.

2.2.2.1 *XIO Bit Calibration*

For the memory interface, only bit (not byte calibration) calibration is done. On the XIO, bit calibration covers the entire 32-bit (or 36-bit if ECC is enabled) memory data channel, as well as the 12-bit command and address interface. Calibration on the XIO must be done at power-on and periodically during system operation.

Table 2-2 shows the basic data structures used in the code. The standard data type of `int` (32 bits) is also used in the code.

1. Both code and data for XIO training reside in the LS of the first functional SPE.

Table 2-2. Data Structures

Data-Structure Definition	Description
`typedef unsigned char uchar;`	An unsigned integer requiring at least 8 bits
`typedef unsigned short uint16;`	An unsigned integer requiring at least 16 bits.
`typedef unsigned int uint32;`	An unsigned integer requiring at least 32 bits.
`typedef unsigned long long uint64;`	An unsigned integer requiring at least 64 bits.

`struct cline_dq {`
`  uint32  data_hi;`
`  uint32  data_lo;`
`  uchar  ecc;`
`};`
	A structure holding 64 bits plus ECC (Error Correction) bits.

`struct cline_dq cline[16];`	A cache line structure containing the data associated with what the memory unit works on normally.

`struct XIO_pin {`
`  int dq_block;`
`  int dq_pin;`
`};`
	A structure to keep track of DQ block and DQ pins inside an XIO. Each XIO contains four DQ blocks and 9 DQ pins.

To perform the initialization of the memory channels, several functions are called. *Table 2-3* on page 59 shows the names and purposes of these functions. The functions are used to make the initialization of the memory subsystem straightforward and to improve readability of the code. The more complicated functions are marked with cross-references to further details later in this chapter.

Table 2-3. Underlying Functions (Page 1 of 3)

Function	Description	Further Details
`mmio_write`	Writes a register inside the MIC MMIO space. Arguments: 1. Address to the MIC register (offset). 2. Bits 0 to 31 of the data to be written to this register. 3. Bits 32 to 63 of the data to be written to this register.	
`mmio_read`	Read a register inside the MIC MMIO space. The value returned is 64 bits. Arguments: 1. Address to the MIC register (offset).	
`mmio_poll`	Continually reads a register until the value returned, ANDed with the mask equals the compare value. Arguments: 1. Address to the MIC register (offset). 2. Bits 0 to 31 of the data mask. 3. Bits 32 to 63 of the data mask. 4. Bits 0 to 31 of the compare value. 5. Bits 32 to 63 of the compare value.	
`mmio_write_xio`	Writes an XIO register inside of a memory channel. Arguments: 1. which memory channel, 0 or 1. 2. which XIO Cell register, 12 bits. 3. Bits 0 to 15 of the desired value to write.	*Section 2.2.2.12* on page 76

Table 2-3. Underlying Functions (Page 2 of 3)

Function	Description	Further Details
`mmio_read_xio`	Reads an XIO register inside of a memory channel. Arguments: 1. Which memory channel, 0 or 1. 2. Which XIO register (12 bits).	*Section 2.2.2.13* on page 76
`mmio_poll_xio`	Continuously reads the XIO register inside of a memory channel until the value return from the register, ANDed with the mask equals the value. Arguments: 1. Which memory channel, 0 or 1. 2. Which XIO register (12 bits). 3. The value to mask (16 bits). 4. The value to compare against (16 bits).	*Section 2.2.2.14* on page 77
`mmio_write_xdram`	Writes the specified XDR DRAM device's register. Arguments: 1. Which memory channel, 0 or 1. 2. Which type of write, which device, which register. 3. The 8 bit value to write.	*Section 2.2.2.15* on page 77
`delay_ns`	Delays the execution of instructions for the amount of time supplied. Arguments: The number of nano-seconds to wait.	
`delay_us`	Delays the execution of instructions for the amount of time supplied. Arguments: The number of micro-seconds to wait.	
`SYSLU_XDR`	Takes the load order double word and maps it to a DQ Pin/Sub-block. Returns a 16 bit result. Arguments: 1. which word (32 bit quantity). 2. programmed width of the device.	*Section 2.2.2.16* on page 78
`SYSLU_MBD`	Creates an XIO_pin structure which it returns. Arguments: 1. which memory channel. 2. which device. 3. which pin.	*Section 2.2.2.17* on page 78
`SYSLU_PAT`	Returns a 16 bit pattern. Arguments: 1. which pattern id. 2. An XIO_Pin structure.	*Section 2.2.2.18* on page 79
`SYSLU_PAT2`	A memory address lookup function, returns the memory address. Arguments: 1. the program width. 2. the pattern index.	*Section 2.2.2.19* on page 80
`WDSL_FMT`	Takes a memory data word and formats it to match the XDR DRAM data buffer implementation. Arguments: The memory word location.	*Section 2.2.2.20* on page 81

Table 2-3. Underlying Functions (Page 3 of 3)

Function	Description	Further Details
`mic_cline_fmt`	Forms a cache line based off from the pattern buffer. Arguments: 1. cache line number. 2. pointer to an array of 16 bit patterns. 3. A cache line structure to be filled in.	*Section 2.2.2.21* on page 81
`mic_pattern_dq_load`	Loads the MIC with the pattern given the number of lines. Arguments: 1. How many patterns to load. 2. A pointer to the pattern structure.	*Section 2.2.2.22* on page 82
`XDR_Store64`	Performs the correct series of commands to create a store of half a cache line (64 bytes). Arguments: The address in memory to cause the store to happen.	*Section 2.2.2.23* on page 83
`XDR_Store128`	Performs the correct series of commands to cause a store on a complete cache line (128 bytes). Arguments: The address in memory to cause the store to happen.	*Section 2.2.2.24* on page 84

The next few lists provide the programming constants needed to read the example code. We assume the `BE_MMIO_Base` registers has been set up through the Configuration Ring with the system-specific memory map for the CBE processor, as described in the Memory Map chapter of the *Cell Broadband Engine Programming Handbook*. If multiple CBE processors are initialized, the base addresses of each CBE processor will be different.

The following MIC definitions refer to registers in the MIC MMIO space:

```
/*MIC Definitions*/
#define BE_MMIO_BASE           0xF0000000
#define MMIO_BE_MIC            (0x50A000 | BE_MMIO_BASE)
#define MIC_CTL_CNFG2          0x040
#define MIC_AUX_TRC_BASE       0x050
#define MIC_AUX_TRC_MAX_ADDR   0x058
#define MIC_AUX_TRC_CUR_ADDR   0x060
#define MIC_AUX_TRC_GRF_ADDR   0x068
#define MIC_AUX_TRC_GRF_DATA   0x070
#define MIC_CTL_CNFG_0         0x080
#define MIC_CALIBRATION_ADDR_0 0x0A0
#define MIC_TM_THRESHOLD_0     0x0A8
#define MIC_QUE_BURSTSIZE_0    0x0B0
#define MIC_DEV_CFG_0          0x0C0
#define MIC_MEM_CFG_0          0x0C8
#define MIC_TRCD_PCHG_0        0x0D0
#define MIC_CMD_DUR_0          0x0D8
#define MIC_CMD_SPC_0          0x0E0
#define MIC_DF_CTL_0           0x0E8
#define MIC_XIO_PTCAL_DATA_0   0x0F0
#define MIC_ECC_ADDR_0         0x0F8
#define YREG_YRAC_DTA_0        0x100
```

```
#define YREG_YDRAM_DTA_0       0x108
#define MIC_YREG_STAT_0        0x110
#define YREG_INIT_CTL_0        0x118
#define YREG_INIT_CNTS_0       0x120
#define MIC_PTCAL_ADR_0        0x130
#define YREG_YRAC_DTA_1        0x140
#define YREG_YDRAM_DTA_1       0x148
#define MIC_YREG_STAT_1        0x150
#define YREG_INIT_CTL_1        0x158
#define YREG_INIT_CNTS_1       0x160
#define MIC_PTCAL_ADR_1        0x170
#define MIC_DEV_CFG_1          0x180
#define MIC_MEM_CFG_1          0x188
#define MIC_TRCD_PCHG_1        0x190
#define MIC_CMD_DUR_1          0x198
#define MIC_CMD_SPC_1          0x1A0
#define MIC_DF_CTL_1           0x1A8
#define MIC_XIO_PTCAL_DATA_1   0x1B0
#define MIC_ECC_ADDR_1         0x1B8
#define MIC_CTL_CNFG_1         0x1C0
#define MIC_CALIBRATION_ADDR_1 0x1E0
#define MIC_TM_THRESHOLD_1     0x1E8
#define MIC_QUE_BURSTSIZE_1    0x1F0
#define MIC_REF_SCB            0x200
#define MIC_EXC                0x208
#define MIC_MNT_CFG            0x210
#define MIC_DF_CONFIG          0x218
#define MIC_FIR                0x230
#define MIC_FIR_DEBUG          0x238
```

The following definitions refer to XIO cell registers used in initialization. These are Rambus XIO registers located in the XIO unit but accessed indirectly through two MIC MMIO registers (YREG_YRAC_DTA_0 and YREG_YRAC_DTA_1), for channel 0 and 1, respectively.

```
/* XIO Cell Registers Used in Initialization */
#define YR_CTL                  0x000
#define CTL_LOCK_STS            0x00
#define CTL_DQ_PLL_ENA          0x06
#define CTL_DQ_DLL_ENA          0x07
#define CTL_PCAL_ACT            0x10
#define CTL_PCAL_CTL            0x11
#define CTL_PCAL_TIMING         0x12
#define CTL_TCAL_RX             0x24
#define CTL_TCAL_TX             0x25
#define CTL_RX_PHASE_MIN        0x26
#define CTL_RX_PHASE_MAX        0x27
#define CTL_TX_PHASE_MIN        0x28
#define CTL_TX_PHASE_MAX        0x29
#define CTL_ITCAL_SAMP          0x2A
```

```
#define YR_RQ_ALL               0x200
#define RQ_SERIAL_CTL           0x01
#define RQ_DLL_CTL              0x02
#define RQ_SWING_OFFSET         0x07
#define RQ_CCAL_VAL             0x09
#define YR_DQ_ALL               0x300
#define DQ_TX_PHASE_CTL         0x0
#define DQ_TX_PHASE             0x1
#define DQ_RX_PHASE_CTL         0x4
#define DQ_RX_PHASE             0x5
#define YR_DQ_GLOBAL            0x0F0
#define DQ_IDAC                 0x1
#define DQ_ECC_CTL              0x2
#define DQ_LPCLK_ADJ            0x4
#define DQ_LPCLK_PHASE          0x5
```

The following definitions refer to Rambus registers in the XIO unit that write data to the XDR DRAMS. These are indirectly accessed through MIC MMIO registers (YREG_YDRAM_DTA_0 and YREG_YDRAM_DTA_1).

```
/* XDR DRAM Registers used in Initialization */
#define XDR_CFG                 0x02  /* Configuration                  */
#define XDR_PM                  0x03  /* Power management               */
#define XDR_WDSL                0x04  /* Write data serial load control */
#define XDR_DLY                 0x1f  /* Delay control                  */
```

The following constants are needed to program the XDR IO Cell:

```
/* XIO Cell Constants (Values) */
#define RX_PHASE_MIN            0x0000
#define RX_PHASE_MAX            0x1400
#define TX_PHASE_MIN            0x1400
#define TX_PHASE_MAX            0x1F00
#define SIMPLE_RX_PHASE_MAX     0x1400
#define SIMPLE_RX_PHASE_MIN     0x0400
#define SIMPLE_TX_PHASE_MIN     0x1200
#define SIMPLE_TX_PHASE_MAX     0x1F00
#define XIO_PCAL_CTL_ENA        0x8000  /* Enable PTCal */
```

These miscellaneous definitions describe the working system using Speed Bin C parts and are used to make the code more readable:

```
/* Miscellaneous Constants */
#define XDR0                    0
#define XDR1                    1
```

```
#define SCMD_SBW                0x1     /* Serial Broadcast Write */
#define SCMD_SDW                0x0     /* Serial Device Write */
#define XDR_NDEV                5       /* Number of device per RQ channel */
#define WBITS                   3       /* device width = 2 ^ WBITS        */
#define XDR_WNATIVE             16      /* Native XDR DRAM width           */
#define PAT_LINES               32      /* Number of cache lines for XDR pattern */
#define NUM_CAL                 128
#define tCWD                    3       /* Use XDR Bin.C */
#define tCAC                    7       /* Use XDR Bin.C */
#define XDR_WPROG               8       /* Programmed XDR DRAM width       */
#define PTRNS_DQ                64
#define XDR_BL                  16      /* Burst length                    */
#define WDSL_BASE_ADDR          0x00000000
#define NUM_POLL                10000
#define tCWD_tCAC               (tCWD << 4) | (tCAC)
#define XDR_NBLK                (XDR_NWR/XDR_NSB)        /* Number of blocks per pattern    */
#define XDR_NSB                 (XDR_WNATIVE/XDR_WPROG)  /* Number of sub blocks            */
#define XDR_PL                  (PAT_LINES*32)           /* XDR DRAM pattern length         */
#define XDR_NWR                 (XDR_PL/XDR_BL)          /* Number of writes per pattern    */
#define XDR_SCMD(CMD, SSID, REG)  ( ((CMD)<<28) | 0x04000000 | ((SSID)<<16) | ((REG)<<8) )
```

To make a meaningful example, this section assumes two working memory channels and an XIO Data Rate of 3.2 Gbps. This means that the reference-clock pins to the XDR DRAM and the XIO (Y0_RQ_CTM, Y1_RQ_CTM) are running at 400 MHz. It also assumes 512 MB of memory.

The initialization of the memory subsystem is done through a variety of steps that are outlined in the *Rambus XDR Initialization Guide (DL-0178)* supplied by Rambus and referred to in the code sample as “XDRIG”. This example is written as though the accesses were to be done through a PPE or SPE.

The initialization is done in eight steps. Step 1 is the initialization and stabilization of the XIO PLL and internal XIO registers, which is done during Phase 2 of the POR sequence (see *Table 2-1 POR Sequence* on page 30). Step 2 starts with the section *Section 2.2.2.3* on page 65. The variable declarations, listed immediately below, are not part of the eight steps.

2.2.2.2 *Variable Declarations*

Some variables are used to store temporary results and have the following definition.

```
/* Variable Declarations */
uint16          calResult;
uint64          data_0, data_1;
uint16          phase_array[2][4][9][2];
int             dqblock;
int             dqpin;
int             reg_address;
int             memory_chn;
int             mem_start = 0;
int             mem_end = 1;
int             block, dev, wd, sb;
```

```
int             xdr_pin;
struct          XIO_pin xio_pin_0, xio_pin_1;
uint16          t, byte10_0, byte10_1, pat_index;
uint16          mc_wd0, mc_wd1;
uint32          mc_addr;
uint64          pt_addr;
uint64          data;
uint16          bit_pattern_dq[PTRNS_DQ * 36]; /* Global Variable with data pattern */
```

2.2.2.3 *Step 2: Initialization of the MIC*

This step occurs before the XIO blocks are initialized and calibrated. It initializes most of the registers in the MIC. In the *Rambus XDR Initialization Guide (DL-0178)*, this step is also known as step 2. (Step 1 is the initialization and stabilization of the XIO PLL and internal XIO registers, done during Phase 2 of the POR sequence as shown in *Table 2-1 POR Sequence* on page 30.)

```
/* MIC Initialization */
mmio_write ( MMIO_BE_MIC | MIC_DEV_CFG_0,          0x48200000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_DEV_CFG_1,          0x48200000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_MEM_CFG_0,          0x00C00000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_MEM_CFG_1,          0x00C00000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_DF_CTL_0,           0x0A543CE0, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_DF_CTL_1,           0x0A543CE0, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_CMD_SPC_0,          0x71841A10, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_CMD_SPC_1,          0x71841A10, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_CMD_DUR_0,          0x5D700000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_CMD_DUR_1,          0x5D700000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_TRCD_PCHG_0,        0x6284055A, 0xD6B00000 );
mmio_write ( MMIO_BE_MIC | MIC_TRCD_PCHG_1,        0x6284055A, 0xD6B00000 );
mmio_write ( MMIO_BE_MIC | MIC_MNT_CFG,            0x7CFE0000, 0x00000000 );
delay_ns   ( 50 ); /* Allow time to enable two-channel configuration of MIC */
mmio_write ( MMIO_BE_MIC | MIC_REF_SCB,            0x06104058, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_FIR,                0x0000FD40, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_FIR_DEBUG,          0x00000280, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_EXC,                0x00000000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_CTL_CNFG_0,         0x80000000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_CTL_CNFG_1,         0x80000000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_DF_CONFIG,          0xF3E40000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_QUE_BURSTSIZE_0,    0x23000000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_QUE_BURSTSIZE_1,    0x23000000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_TM_THRESHOLD_0,     0x09127754, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_TM_THRESHOLD_1,     0x09127754, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_CTL_CNFG2,          0x12000000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_AUX_TRC_BASE,       0x00000000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_AUX_TRC_MAX_ADDR,   0x0000000F, 0xFFFFFF80 );

/* make sure reg reset has dropped */
mmio_poll  ( MMIO_BE_MIC | MIC_YREG_STAT_0,
             0x00001000, 0x00000000,   /* mask, check this bit */
```

```
                0x00000000, 0x00000000 ); /* value, must match this value */
```

2.2.2.4 *Step 3: XIO Initialization*

The next step is to enable and configure the XIO and calibrate the I/O cells on the CBE processor. In the *Rambus XDR Initialization Guide (DL-0178)* (XDRIG), this step is known as step 3.

```
/* Enable DQ Block PLLs/DLL (XDRIG 3.1) */
/*Initialize RQ driver current*/
mmio_write_xio (0, (YR_RQ_ALL | RQ_CCAL_VAL), 0x0040);
mmio_write_xio (1, (YR_RQ_ALL | RQ_CCAL_VAL), 0x0040);

/*Enable DQ Block PLLs */
mmio_write_xio (0, (YR_CTL | CTL_DQ_PLL_ENA), 0x01ff);
mmio_write_xio (1, (YR_CTL | CTL_DQ_PLL_ENA), 0x01ff);

/*Poll for DQ PLLs locked */
mmio_poll_xio (0, (YR_CTL | CTL_LOCK_STS), 0x0002, 0x0002);
mmio_poll_xio (1, (YR_CTL | CTL_LOCK_STS), 0x0002, 0x0002);

/*Enable DQ LPCLK DLLs*/
mmio_write_xio (0, (YR_CTL | CTL_DQ_DLL_ENA), 0x01ff);
mmio_write_xio (1, (YR_CTL | CTL_DQ_DLL_ENA), 0x01ff);

/*Poll DQ PLLs/DLLs*/
mmio_poll_xio (0, (YR_CTL | CTL_LOCK_STS), 0x0006, 0x0006);
mmio_poll_xio (1, (YR_CTL | CTL_LOCK_STS), 0x0006, 0x0006);

/*Set up LP_CYC*/
mmio_write_xio (0, (YR_DQ_ALL | YR_DQ_GLOBAL | DQ_LPCLK_ADJ), 0xC438);
mmio_write_xio (1, (YR_DQ_ALL | YR_DQ_GLOBAL | DQ_LPCLK_ADJ), 0xC438);
mmio_write_xio (0, (YR_DQ_ALL | YR_DQ_GLOBAL | DQ_LPCLK_PHASE), 0x8000);
mmio_write_xio (1, (YR_DQ_ALL | YR_DQ_GLOBAL | DQ_LPCLK_PHASE), 0x8000);


/* Enable RQ Block DLL (XDRIG 3.2) */
/*Poll RQ_QCLK_LOCK_ALL */
mmio_poll_xio (0, (YR_CTL | CTL_LOCK_STS), 0x0007, 0x0007);
mmio_poll_xio (1, (YR_CTL | CTL_LOCK_STS), 0x0007, 0x0007);

/*Set REFS, latch PCLK phase and relock dll*/
mmio_write_xio (0, (YR_RQ_ALL | RQ_DLL_CTL), 0x0009);
mmio_write_xio (1, (YR_RQ_ALL | RQ_DLL_CTL), 0x0009);

/*Poll RQ_QCLK_LOCK_ALL */
mmio_poll_xio (0, (YR_CTL | CTL_LOCK_STS), 0x0007, 0x0007);
mmio_poll_xio (1, (YR_CTL | CTL_LOCK_STS), 0x0007, 0x0007);
```

```
/*Set skip latch */
mmio_write_xio (0, (YR_RQ_ALL | RQ_DLL_CTL), 0x000B);
mmio_write_xio (1, (YR_RQ_ALL | RQ_DLL_CTL), 0x000B);


/* Set RCLK_ENA and TCLK_ENA (XDRIG 3.3) */
mmio_write (MMIO_BE_MIC | YREG_INIT_CNTS_0, 0xC8000000, 0x00000000);
mmio_write (MMIO_BE_MIC | YREG_INIT_CNTS_1, 0xC8000000, 0x00000000);


/* Initial RQ CCAL (XDRIG 3.4) */
/*Set RQ_SWING_OFFSET*/
mmio_write_xio (0, (YR_RQ_ALL | RQ_SWING_OFFSET), 0x0000);
mmio_write_xio (1, (YR_RQ_ALL | RQ_SWING_OFFSET), 0x0000);

/*Initial RQ Current Calibration (CCAL)*/
for (int n = 0; n < NUM_CAL; n += 1) {
   mmio_write_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0011);
   mmio_write_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0011);

   mmio_poll_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0020, 0x0020);
   mmio_poll_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0020, 0x0020);

   mmio_write_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0000);
   mmio_write_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0000);
}


/* Initial DQ ZCAL (XDRIG 3.5) */
for (int n = 0; n < NUM_CAL; n += 1)  {
   mmio_write_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0010);
   mmio_write_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0010);
   mmio_poll_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0020, 0x0020);
   mmio_poll_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0020, 0x0020);

   mmio_write_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0000);
   mmio_write_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0000);
}


/* Set DQ IDAC Value (XDRIG 3.6) and XIO Register Configuration (XDRIG 3.7) */
mmio_write_xio (0, (YR_DQ_ALL | YR_DQ_GLOBAL | DQ_IDAC), 0x0040);
mmio_write_xio (1, (YR_DQ_ALL | YR_DQ_GLOBAL | DQ_IDAC), 0x0040);

mmio_write_xio (0, (YR_DQ_ALL | YR_DQ_GLOBAL | DQ_ECC_CTL), 0x0001);
mmio_write_xio (1, (YR_DQ_ALL | YR_DQ_GLOBAL | DQ_ECC_CTL), 0x0001);
```

2.2.2.5 *Step 4: XDR DRAM Initialization*

The next step is to initialize the XDR DRAM chips that are external to the CBE processor. In the *Rambus XDR Initialization Guide (DL-0178)* (XDRIG) this is known as step 4.

```
/* Reset and Serial ID Assignment (XDRIG 4.1) */
mmio_write_xio (0, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0001);
mmio_write_xio (1, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0001);

for (int n = 0; n < 4; n += 1) {
   mmio_write_xio (0, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0003);
   mmio_write_xio (1, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0003);
   mmio_write_xio (0, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0001);
   mmio_write_xio (1, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0001);
}

mmio_write_xio (0, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0000);
mmio_write_xio (1, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0000);

for (int n = 0; n < XDR_NDEV; n += 1) {
   mmio_write_xio (0, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0002);
   mmio_write_xio (1, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0002);
   mmio_write_xio (0, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0000);
   mmio_write_xio (1, (YR_RQ_ALL | RQ_SERIAL_CTL), 0x0000);
}


/* XDR DRAM Register Configuration (XDRIG 4.2) */
/* Set XDR device width to limit VDDIO power at device turn-on */
mmio_write_xdram (0, XDR_SCMD (SCMD_SBW, 0, XDR_CFG), WBITS);
mmio_write_xdram (1, XDR_SCMD (SCMD_SBW, 0, XDR_CFG), WBITS);

/*  set tCWD_tCAC */
mmio_write_xdram (0, XDR_SCMD (SCMD_SBW, 0, XDR_DLY), tCWD_tCAC);
mmio_write_xdram (1, XDR_SCMD (SCMD_SBW, 0, XDR_DLY), tCWD_tCAC);


/* XDR Power-down Exit (XDRIG 4.3) */
mmio_write_xdram (0, XDR_SCMD (SCMD_SBW, 0, XDR_PM), 0x01);
mmio_write_xdram (1, XDR_SCMD (SCMD_SBW, 0, XDR_PM), 0x01);
delay_ns (10240);   /* Meet power-down requirements of XDR DRAMs */


/* XDR Bank Conditioning (XDRIG 4.4) */
for (int n = 0; n < 8; n += 1) {
   mmio_write (MMIO_BE_MIC | MIC_EXC, 0x06000000, 0x00000000);
   mmio_poll (MMIO_BE_MIC | MIC_EXC, 0x04000000, 0x00000000, 0x00000000, 0x00000000);
}


/* MC Refresh Enable (XDRIG 4.5) */
```

```
mmio_write (MMIO_BE_MIC | MIC_EXC, 0x40000000, 0x00000000);


/* XDR Initial ZCAL/CCAL (XDRIG 4.5) */
for (int n = 0; n < NUM_CAL; n += 1)  {
   mmio_write_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0012);
   mmio_write_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0012);

   mmio_poll_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0020, 0x0020);
   mmio_poll_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0020, 0x0020);

   mmio_write_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0000);
   mmio_write_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0000);
}

for (int n = 0; n < NUM_CAL; n += 1) {
   mmio_write_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0013);
   mmio_write_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0013);

   mmio_poll_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0020, 0x0020);
   mmio_poll_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0020, 0x0020);

   mmio_write_xio (0, (YR_CTL | CTL_PCAL_ACT), 0x0000);
   mmio_write_xio (1, (YR_CTL | CTL_PCAL_ACT), 0x0000);
}
```

2.2.2.6 *Step 5.1: XDR DRAM Load*

After the MIC, XIO cells, and XDR DRAMS are initialized and partially calibrated, the initialization process proceeds with performing timing calibrations. The first step is to load a pattern into the XDR DRAMs, as shown below, and into the MIC as shown in *Section 2.2.2.7* on page 70. This step is known in the *Rambus XDR Initialization Guide (DL-0178)* (XDRIG) as step 5.

```
/* Load up the XDR DRAM serially, 8 bits at a time */
/* Set XDR DRAM SLE (Serial Load Enable) bit in all XDR DRAMs */
mmio_write_xdram (0, XDR_SCMD (SCMD_SBW, 0, XDR_CFG), 0x10 | WBITS);
mmio_write_xdram (1, XDR_SCMD (SCMD_SBW, 0, XDR_CFG), 0x10 | WBITS);

for (block = 0; block < XDR_NBLK; block++) {
   for (dev = 0; dev < XDR_NDEV; dev++) {
      for (wd = 0; wd < XDR_WNATIVE; wd++) {
           t = SYSLU_XDR (wd, XDR_WPROG);
           xdr_pin = (t & 0x0f00) >> 8;
           sb = t & 0x00ff;
           xio_pin_0 = SYSLU_MBD (0, dev, xdr_pin);
           xio_pin_1 = SYSLU_MBD (1, dev, xdr_pin);

           pat_index = (block & 0x3e) * XDR_NSB + 2 * sb + (block & 0x01);
           mc_wd0 = SYSLU_PAT (pat_index, xio_pin_0);
```

```
            mc_wd1 = SYSLU_PAT (pat_index, xio_pin_1);
            byte10_0 = WDSL_FMT (mc_wd0);
            byte10_1 = WDSL_FMT (mc_wd1);

          mmio_write_xdram (XDR0, XDR_SCMD (SCMD_SDW, dev, XDR_WDSL), (byte10_0 & 0xff00) >> 8);
          mmio_write_xdram (XDR1, XDR_SCMD (SCMD_SDW, dev, XDR_WDSL), (byte10_1 & 0xff00) >> 8);
          mmio_write_xdram (XDR0, XDR_SCMD (SCMD_SDW, dev, XDR_WDSL), byte10_0 & 0x00ff);
          mmio_write_xdram (XDR1, XDR_SCMD (SCMD_SDW, dev, XDR_WDSL), byte10_1 & 0x00ff);
      }
    }
    /* read MIC_YREG_STAT to verify completion of the last mmio_write_xdram command */
    mmio_read (MMIO_BE_MIC | MIC_YREG_STAT_0);
    mmio_read (MMIO_BE_MIC | MIC_YREG_STAT_1);

    for (sb = 0; sb < XDR_NSB; sb++)     {
       pat_index = (block & 0x3e) * XDR_NSB + 2 * sb + (block & 0x01);
       mc_addr = WDSL_BASE_ADDR | SYSLU_PAT2 (XDR_WPROG, pat_index);
       XDR_store64 (mc_addr);    /* XDR 0 */
       XDR_store64 (mc_addr + 0x80);     /* XDR 1 */
    }
}
/* Check to see if the store queue is empty */
mmio_poll (MMIO_BE_MIC | MIC_YREG_STAT_0, 0x00000400, 0x00000000, 0x00000400, 0x00000000);
mmio_poll (MMIO_BE_MIC | MIC_YREG_STAT_1, 0x00000400, 0x00000000, 0x00000400, 0x00000000);

/* Reset SLE bit in XDR DRAM Configuration register */
mmio_write_xdram (0, XDR_SCMD (SCMD_SBW, 0, XDR_CFG), WBITS);
mmio_write_xdram (1, XDR_SCMD (SCMD_SBW, 0, XDR_CFG), WBITS);

/* verify the SLE bit write has completed */
mmio_read (MMIO_BE_MIC | MIC_YREG_STAT_0);
mmio_read (MMIO_BE_MIC | MIC_YREG_STAT_1);
```

2.2.2.7 *Step 5.2: XDR MIC Pattern Load*

Details of the MIC-pattern load are found in the support function, `mic_pattern_dq_load`, as described in *Section 2.2.2.22* on page 82. The MIC-pattern load also includes the loading of the periodic timing-calibration pattern space.This step is known in the *Rambus XDR Initialization Guide (DL-0178)* (XDRIG) as step 5.

```
/* Code that does the pattern load of the MIC and sets up periodic timing calibration
pattern space.*/
/* This calls the function that does all of the pattern load*/
mic_pattern_dq_load (PAT_LINES, bit_pattern_dq);

pt_addr = 0x7800000;/* start at address 120M in memory */

pt_addr = pt_addr & 0x0000000FFFFFFF00ULL;/* get the important 28 bits of address */
```

```
/* set up the address for periodic timing calibration */
pt_addr = pt_addr << 28;/* move msb to bit 0 */
mmio_write (MMIO_BE_MIC | MIC_PTCAL_ADR_0, (pt_addr >> 32) & 0xFFFFFFFF, pt_addr &
0xFFFFFFFF);
mmio_write (MMIO_BE_MIC | MIC_PTCAL_ADR_1, (pt_addr >> 32) & 0xFFFFFFFF, pt_addr &
0xFFFFFFFF);

/* set up the special pattern to be stored in the pattern array */
data = mmio_read (MMIO_BE_MIC | MIC_DF_CONFIG);

/* set 16:17 of df config to be "11" so we can write Slot C and selects */
mmio_write (MMIO_BE_MIC | MIC_DF_CONFIG,
    ((data >> 32) & 0xFFFF3FFF) | 0x0000C000, data & 0xFFFFFFFF);
mmio_write (MMIO_BE_MIC | MIC_XIO_PTCAL_DATA_0, 0x5349ACB6, 0x88C46220);
mmio_write (MMIO_BE_MIC | MIC_XIO_PTCAL_DATA_1, 0x5349ACB6, 0x88C46220);

/* now reset df config bits 16:17 and write slot A and B */
mmio_write (MMIO_BE_MIC | MIC_DF_CONFIG, (data >> 32) & 0xFFFF3FFF,
          data & 0xFFFFFFFF);
mmio_write (MMIO_BE_MIC | MIC_XIO_PTCAL_DATA_0, 0xEDD61229, 0x594BA6B4);
mmio_write (MMIO_BE_MIC | MIC_XIO_PTCAL_DATA_1, 0xEDD61229, 0x594BA6B4);
```

2.2.2.8 *Step 6: Initial RX Timing Calibration*

The next step is to perform the initial receive (RX) timing calibration, also known as step 6 in the *Rambus XDR Initialization Guide (DL-0178)* (XDRIG).

```
/* XIO Register Setup (XDRIG 6.1) */
mmio_write_xio (0, (YR_CTL | CTL_RX_PHASE_MIN), RX_PHASE_MIN);
mmio_write_xio (1, (YR_CTL | CTL_RX_PHASE_MIN), RX_PHASE_MIN);

mmio_write_xio (0, (YR_CTL | CTL_RX_PHASE_MAX), RX_PHASE_MAX);
mmio_write_xio (1, (YR_CTL | CTL_RX_PHASE_MAX), RX_PHASE_MAX);

mmio_write_xio (0, (YR_CTL | CTL_ITCAL_SAMP), 0x0005);
mmio_write_xio (1, (YR_CTL | CTL_ITCAL_SAMP), 0x0005);

mmio_write_xio (0, (YR_CTL | CTL_TCAL_RX), 0x0007);
mmio_write_xio (1, (YR_CTL | CTL_TCAL_RX), 0x0007);


/* Poll on TCAL Done (XDRIG 6.2) */
mmio_poll_xio (0, (YR_CTL | CTL_TCAL_RX), 0x0008, 0x0008);
mmio_poll_xio (1, (YR_CTL | CTL_TCAL_RX), 0x0008, 0x0008);



/* Check Results and Clear Timing Calibration Enable (TCEN) (XDRIG 6.3) */
```

```
calResult = mmio_read_xio (0, (YR_CTL | CTL_TCAL_RX));
if ((0x0100 & calResult) == 0x0100) {
   printf ("\n\n XDR0 RX TCAL failed. result=0x%04X \n\n", calResult);
}

calResult = mmio_read_xio (1, (YR_CTL | CTL_TCAL_RX));
if ((0x0100 & calResult) == 0x0100) {
   printf ("\n\n XDR1 RX TCAL failed. result=0x%04X \n\n", calResult);
}

mmio_write_xio (0, (YR_CTL | CTL_TCAL_RX), 0x0000);
mmio_write_xio (1, (YR_CTL | CTL_TCAL_RX), 0x0000);
```

2.2.2.9 *Step 7: Initial TX Timing Calibration*

This is where the transmit timing is calibrated, corresponding to step 7 of the *Rambus XDR Initialization Guide (DL-0178)* (XDRIG). At the end of this step, the MIC is taken out of its initialization mode.

```
/* XIO Register Setup (XDRIG 7.1) */
mmio_write_xio (0, (YR_CTL | CTL_TX_PHASE_MIN), TX_PHASE_MIN);
mmio_write_xio (1, (YR_CTL | CTL_TX_PHASE_MIN), TX_PHASE_MIN);

mmio_write_xio (0, (YR_CTL | CTL_TX_PHASE_MAX), TX_PHASE_MAX);
mmio_write_xio (1, (YR_CTL | CTL_TX_PHASE_MAX), TX_PHASE_MAX);

mmio_write_xio (0, (YR_CTL | CTL_ITCAL_SAMP), 0x0005);
mmio_write_xio (1, (YR_CTL | CTL_ITCAL_SAMP), 0x0005);

mmio_write_xio (0, (YR_CTL | CTL_TCAL_TX), 0x0007);
mmio_write_xio (1, (YR_CTL | CTL_TCAL_TX), 0x0007);

/* Poll on TCAL Done (XDRIG 7.2) */
mmio_poll_xio (0, (YR_CTL | CTL_TCAL_TX), 0x0008, 0x0008);
mmio_poll_xio (1, (YR_CTL | CTL_TCAL_TX), 0x0008, 0x0008);


/* Check Results and Clear Timing Calibration Enable (TCEN) (XDRIG 7.3) */
calResult = mmio_read_xio (0, (YR_CTL | CTL_TCAL_TX));
if ((0x0100 & calResult) == 0x0100)
   printf ("\n\n XIO0 TX TCAL failed. result=0x%04X \n\n", calResult);
calResult = mmio_read_xio (1, (YR_CTL | CTL_TCAL_TX));
if ((0x0100 & calResult) == 0x0100)
   printf ("\n\n XIO1 TX TCAL failed. result=0x%04X \n\n", calResult);

mmio_write_xio (0, (YR_CTL | CTL_TCAL_TX), 0x0000);
mmio_write_xio (1, (YR_CTL | CTL_TCAL_TX), 0x0000);
```

```
/* Get the MIC out of Initialization Mode. Allow completion of up to 15 refreshes,
which may have been deferred during calibration due to calibration pattern execution
*/
delay_ns (3450); // let deferred refreshes finish 2.5 ns * 60 * 15 * 1.5, worst case

mmio_write (MMIO_BE_MIC | YREG_INIT_CTL_0, 0x00200000, 0x00000000);
delay_ns (100);
mmio_write (MMIO_BE_MIC | YREG_INIT_CTL_1, 0x00200000, 0x00000000);
delay_ns (100);

delay_ns (360);   /* Give MIC time to invalidate pattern buffers */
data_0 = mmio_read (MMIO_BE_MIC | MIC_TM_THRESHOLD_0);
data_1 = mmio_read (MMIO_BE_MIC | MIC_TM_THRESHOLD_1);
mmio_write (MMIO_BE_MIC | MIC_TM_THRESHOLD_0, (data_0 >> 32) | 0x2, data_0 &
0xFFFFFFFF);
mmio_write (MMIO_BE_MIC | MIC_TM_THRESHOLD_1, (data_1 >> 32) | 0x2, data_1 &
0xFFFFFFFF);

delay_ns (200);  /* Allow threshold values with correct number of commands in MIC */
XDR_store128 (0);
delay_ns (200);   /* Allow time for store to complete */
XDR_store128 (128);
delay_ns (1000);  /* Allow time for store to complete */

mmio_read (MMIO_BE_MIC | MIC_ECC_ADDR_0);
mmio_read (MMIO_BE_MIC | MIC_ECC_ADDR_1);
/* Enable Power Saving Mode */
mmio_write (MMIO_BE_MIC | MIC_CTL_CNFG2, 0x10000000, 0x00000000); //bit 7 set to 0
mmio_write (MMIO_BE_MIC | MIC_CTL_CNFG_0, 0x00000000, 0x00000000); //bit 1 set to 0
mmio_write (MMIO_BE_MIC | MIC_CTL_CNFG_1, 0x00000000, 0x00000000); // bit 1 set to 0
```

2.2.2.10 *Step 8: Second-Pass Simple Timing Calibration*

The next step is to perform simple timing calibration. This corresponds to step 8 of the *Rambus XDR Initialization Guide (DL-0178)* (XDRIG). This process requires a small amount of storage to retain and restore the calibrated center phases of the RX and TX calibration results, for each pin in each block on each memory channel.

```
/* Save Complex Center Phases (XDRIG 8.1) */
for (dqblock = 0; dqblock < 4; dqblock++)  {
   for (dqpin = 0; dqpin < 9; dqpin++) {
      reg_address = 0x800 | (dqblock << 8) | (dqpin << 4);
      for (memory_chn = mem_start; memory_chn <= mem_end;   memory_chn++)
            phase_array[memory_chn][dqblock][dqpin][0] =
             mmio_read_xio (memory_chn, (reg_address | DQ_RX_PHASE));
      for (memory_chn = mem_start; memory_chn <= mem_end;   memory_chn++)
            phase_array[memory_chn][dqblock][dqpin][1] =
            mmio_read_xio (memory_chn, (reg_address | DQ_TX_PHASE));
    }
```

```
}


/* Simple RX TCal (XDRIG 8.2) */
mmio_write_xio (0, (YR_CTL | CTL_RX_PHASE_MIN), SIMPLE_RX_PHASE_MIN);
mmio_write_xio (1, (YR_CTL | CTL_RX_PHASE_MIN), SIMPLE_RX_PHASE_MIN);
mmio_write_xio (0, (YR_CTL | CTL_RX_PHASE_MAX), SIMPLE_RX_PHASE_MAX);
mmio_write_xio (1, (YR_CTL | CTL_RX_PHASE_MAX), SIMPLE_RX_PHASE_MAX);
mmio_write_xio (0, (YR_CTL | CTL_ITCAL_SAMP), 0x0001);
mmio_write_xio (1, (YR_CTL | CTL_ITCAL_SAMP), 0x0001);
mmio_write_xio (0, (YR_CTL | CTL_TCAL_RX), 0x0006);
mmio_write_xio (1, (YR_CTL | CTL_TCAL_RX), 0x0006);

mmio_poll_xio (0, (YR_CTL | CTL_TCAL_RX), 0x0008, 0x0008);
mmio_poll_xio (1, (YR_CTL | CTL_TCAL_RX), 0x0008, 0x0008);

calResult = mmio_read_xio (0, (YR_CTL | CTL_TCAL_RX));
if ((0x0100 & calResult) == 0x0100)
  printf("\n\nXDR0 RX TCAL failed. result=0x%04X \n\n", calResult);

calResult = mmio_read_xio (1, (YR_CTL | CTL_TCAL_RX));
if ((0x0100 & calResult) == 0x0100)
  printf  ("\n\nXDR1 RX TCAL failed. result=0x%04X \n\n", calResult);

mmio_write_xio (0, (YR_CTL | CTL_TCAL_RX), 0x0000);
mmio_write_xio (1, (YR_CTL | CTL_TCAL_RX), 0x0000);


/* Simple TX TCal (XDRIG 8.3) */
mmio_write_xio (0, (YR_CTL | CTL_TX_PHASE_MIN), SIMPLE_TX_PHASE_MIN);
mmio_write_xio (1, (YR_CTL | CTL_TX_PHASE_MIN), SIMPLE_TX_PHASE_MIN);
mmio_write_xio (0, (YR_CTL | CTL_TX_PHASE_MAX), SIMPLE_TX_PHASE_MAX);
mmio_write_xio (1, (YR_CTL | CTL_TX_PHASE_MAX), SIMPLE_TX_PHASE_MAX);
mmio_write_xio (0, (YR_CTL | CTL_ITCAL_SAMP), 0x0001);
mmio_write_xio (1, (YR_CTL | CTL_ITCAL_SAMP), 0x0001);

mmio_write_xio (0, (YR_CTL | CTL_TCAL_TX), 0x0006);    /* Begin TX phase sweep */
mmio_write_xio (1, (YR_CTL | CTL_TCAL_TX), 0x0006);

mmio_poll_xio (0, (YR_CTL | CTL_TCAL_TX), 0x0008, 0x0008);
mmio_poll_xio (1, (YR_CTL | CTL_TCAL_TX), 0x0008, 0x0008);

calResult = mmio_read_xio (0, (YR_CTL | CTL_TCAL_TX));
if ((0x0100 & calResult) == 0x0100)
   printf("\n\nXI00 TX TCAL failed. result=0x%04X \n\n", calResult);

calResult = mmio_read_xio (1, (YR_CTL | CTL_TCAL_TX));
if ((0x0100 & calResult) == 0x0100)
   printf  ("\n\nXI01 TX TCAL failed. result=0x%04X \n\n", calResult);
```

```
mmio_write_xio (0, (YR_CTL | CTL_TCAL_TX), 0x0000);
mmio_write_xio (1, (YR_CTL | CTL_TCAL_TX), 0x0000);


/* Restore Complex Center Phases (XDRIG 8.4) */
for (dqblock = 0; dqblock < 4; dqblock++) {
   for (dqpin = 0; dqpin < 9; dqpin++) {
      reg_address = 0x800 | (dqblock << 8) | (dqpin << 4);
      for (memory_chn = mem_start; memory_chn <= mem_end;  memory_chn++)
        mmio_write_xio (memory_chn, (reg_address | DQ_RX_PHASE),
              phase_array[memory_chn][dqblock][dqpin][0]);
      for (memory_chn = mem_start; memory_chn <= mem_end; memory_chn++)
        mmio_write_xio (memory_chn, (reg_address | DQ_TX_PHASE),
              phase_array[memory_chn][dqblock][dqpin][1]);
   }
}
```

2.2.2.11 *Step 9: Enable Periodic Calibration and Additional MIC Configurations*

This corresponds to step 9 of the *Rambus XDR Initialization Guide (DL-0178)* (XDRIG), with some additions. The beginning of this phase finishes the initialization of the MIC and clears memory. During this phase and after the operating system is running, allocation of addresses for periodic timing calibration (PCAL), requiring at least one cache line per memory channel[1], is needed. This allocation, usually done by the operating system, will provide a free memory location for these periodic timing calibrations to be performed.

```
/* Clear Error bits, turn on ECC, and enable other modes in the MIC*/
mmio_write ( MMIO_BE_MIC | MIC_FIR,              0x0000FD7C, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_FIR_DEBUG,        0x00000280, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_DF_CONFIG,        0x32640000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_CTL_CNFG2,        0x10000000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_CTL_CNFG_0,       0x00000000, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_CTL_CNFG_1,       0x00000000, 0x00000000 );

mmio_write (MMIO_BE_MIC | MIC_EXC, 0xC0000000, 0x00000000);
delay_us (1000);    /* Allow time for zeroing of memory */
mmio_poll (MMIO_BE_MIC | MIC_EXC, 0x80000000, 0x00000000, 0x00000000,  0x00000000);
```

At this point, memory has been zeroed with good ECC. Operating system can now be loaded into memory. Once a memory location has been picked and reserved (in this case a region from 0xBE1100 to 0xBF1100 is selected), the following code is run.

```
/* Set up a memory location that is free for periodic timing calibration */
```

1. It may not be possible for an operating system to allocate just one cache line for periodic timing calibration. In reality, at least eight cache lines need to be allocated due to the restrictions in the MIC_Calibration_Addr_n register, although only one cache line is used.

```
/* These registers are set to point to memory where periodic timing
   Calibrations can be performed, usually done before step 9 and once the
   operating system is functional. These registers need to set with
   correct values based on the system memory map */
mmio_write ( MMIO_BE_MIC | MIC_PTCAL_ADR_0,        0x000BE110, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_PTCAL_ADR_1,        0x000BE110, 0x00000000 );
mmio_write ( MMIO_BE_MIC | MIC_CALIBRATION_ADDR_0, 0x8005F088, 0x005F8880 );
mmio_write ( MMIO_BE_MIC | MIC_CALIBRATION_ADDR_1, 0x8005F088, 0x005F8880 );


/* Enable Periodic Calibration (PCAL) (XDRIG 8.1) and enable scrubbing */
mmio_write_xio (0, (YR_CTL | CTL_PCAL_TIMING), 0xFC0A);
mmio_write_xio (1, (YR_CTL | CTL_PCAL_TIMING), 0xFC0A);

mmio_write_xio (0, (YR_CTL | CTL_PCAL_CTL), XIO_PCAL_CTL_ENA);
mmio_write_xio (1, (YR_CTL | CTL_PCAL_CTL), XIO_PCAL_CTL_ENA);

/*enable Scrubbing*/
mmio_write (MMIO_BE_MIC | MIC_EXC, 0x60000000, 0x00000000);
```

2.2.2.12 *Support Functions: mmio_write_xio*

The following functions, from this section to the end of the chapter, are used during the initialization described in earlier sections of the chapter.

```
/* mmio_write_xio */
int mmio_write_xio(int xio, uint16 xio_addr, uint16 data){
  if (xio == XDR0)  {
    mmio_write(MMIO_BE_MIC | YREG_YRAC_DTA_0,  (xio_addr << 16) | (data & 0xffff), 0);
  }  else {
    mmio_write(MMIO_BE_MIC | YREG_YRAC_DTA_1, (xio_addr << 16) | (data & 0xffff), 0);
  }
  return 0;
}
```

2.2.2.13 *Support Functions: mmio_read_xio*

```
/*mmio_read_xio*/
uint16 mmio_read_xio (int xio, uint16 xio_addr) {
  uint64 data;
  if (xio == XDR0)  {
    mmio_write(MMIO_BE_MIC | YREG_YRAC_DTA_0,  0x10000000 | (xio_addr<<16), 0);
    data = mmio_read(MMIO_BE_MIC | YREG_YRAC_DTA_0);
  }  else  {
    mmio_write(MMIO_BE_MIC | YREG_YRAC_DTA_1,  0x10000000 | (xio_addr<<16), 0);
    data = mmio_read(MMIO_BE_MIC | YREG_YRAC_DTA_1);
  }
```

```
  return (uint16)((data >> 32) & 0xFFFF);
}
```

2.2.2.14 *Support Functions: mmio_poll_xio*

```
/* mmio_poll_xio */
int mmio_poll_xio (int xio , int xio_addr, uint16 mask, uint16 test)
{
  int n = NUM_POLL;
  uint64 data;
  if (xio == XDR0)  {
    mmio_write(MMIO_BE_MIC | YREG_YRAC_DTA_0,  0x10000000 | (xio_addr<<16), 0);
    data = mmio_read(MMIO_BE_MIC | YREG_YRAC_DTA_0);
  }  else  {
    mmio_write(MMIO_BE_MIC | YREG_YRAC_DTA_1,  0x10000000 | (xio_addr<<16), 0);
    data = mmio_read(MMIO_BE_MIC | YREG_YRAC_DTA_1);
  }

  while ((data & mask) != (mask & test))
  {
    delay_us(1000);
    if (xio == XDR0)  {
        data = mmio_read(MMIO_BE_MIC | YREG_YRAC_DTA_0);
    }  else  {
        data = mmio_read(MMIO_BE_MIC | YREG_YRAC_DTA_1);
    }
    if (--n == 0)     {
      printf("mmio_poll_xio : Poll failed.  Address=%08x, Mask=%04x, data=%04X\n",
             xio_addr, mask, data);
    }
  }
  return 0;
}
```

2.2.2.15 *Support Functions: mmio_write_xdram*

```
/* mmio_write_xdram */
/* MIC XDR DRAM access accelerator:      */
/*  bit 0:1   - Reserved                 */
/*  bit 2:3   - SCMD                     */
/*  bit 4:15  - XIO_Address              */
/*  bit 16:17 - Reserved, drive 0        */
/*  bit 18:23 - SID                      */
/*  bit 24:31 - Data                     */

/* mmio_write_xdram: Write a XDR DRAM value using the MIC serial bus assist */
int mmio_write_xdram(int xio, uint32 address, uint16 data) {
```

```
  if (xio == XDRO) {
     mmio_write(MMIO_BE_MIC | YREG_YDRAM_DTA_0,  address | (data & 0x00ff), 0);
  }  else {
     mmio_write(MMIO_BE_MIC | YREG_YDRAM_DTA_1,  address | (data & 0x00ff), 0);
  }
  return 0;
}
```

2.2.2.16 *Support Functions: SYSLU_XDR*

```
/* SYSLU_XDR: XDR DRAM WDSL word load order to DQ Pin/Sub-block index map */
/*            DQ Pin returned in high byte                                */
/*            Sub-Block returned in low byte                              */
uint16 SYSLU_XDR (int wd, int WProg) {

  uint16 x16[16] = { 0x0F00, 0x0700, 0x0B00, 0x0300,
    0x0D00, 0x0500, 0x0900, 0x0100,
    0x0000, 0x0800, 0x0400, 0x0C00,
    0x0200, 0x0A00, 0x0600, 0x0E00
  };

  uint16 x8[16] = { 0x0701, 0x0700, 0x0301, 0x0300,
    0x0501, 0x0500, 0x0101, 0x0100,
    0x0000, 0x0001, 0x0400, 0x0401,
    0x0200, 0x0201, 0x0600, 0x0601
  };

  uint16 x4[16] = { 0x0303, 0x0301, 0x0302, 0x0300,
    0x0103, 0x0101, 0x0102, 0x0100,
    0x0000, 0x0002, 0x0001, 0x0003,
    0x0200, 0x0202, 0x0201, 0x0203
  };

  if (WProg == 16)
    return x16[wd];
  else if (WProg == 8)
    return x8[wd];
  else if (WProg == 4)
    return x4[wd];
  else {
      printf ("SYSLU_XDR: invalid value for WProg\n");
      return 0;
    }
}
```

2.2.2.17 *Support Functions: SYSLU_MBD*

```
/* SYSLU_MBD*/
struct XIO_pin SYSLU_MBD (int channel, int device, int pin) {
  struct XIO_pin result;

  int xio_block_lu[2][5][8] = { {{0, 3, 99, 99, 99, 99, 0, 3},  /* channel 0 */
                                 {1, 3, 0, 3, 0, 3, 0, 3},
                                 {1, 2, 1, 2, 1, 3, 1, 2},
                                 {0, 2, 0, 3, 0, 3, 0, 2},
                                 {1, 2, 1, 2, 1, 2, 1, 2}},

  {{0, 3, 99, 99, 99, 99, 0, 3},        /* channel 1 */
   {0, 2, 0, 3, 0, 3, 0, 3},
   {1, 2, 1, 3, 1, 2, 1, 2},
   {1, 2, 0, 3, 0, 3, 1, 3},
   {1, 2, 0, 2, 1, 2, 1, 2}}
  };


  int xio_pin_lu[2][5][8] = { {{2, 8, 99, 99, 99, 99, 8, 6},    /* channel 0 */
                               {0, 3, 3, 5, 7, 7, 1, 0},
                               {8, 2, 1, 7, 7, 1, 3, 8},
                               {0, 0, 4, 4, 5, 2, 6, 6},
                               {6, 1, 2, 3, 4, 5, 5, 4}},

  {{8, 7, 99, 99, 99, 99, 4, 8},        /* channel 1 */
   {7, 7, 5, 3, 1, 0, 2, 6},
   {3, 8, 1, 1, 0, 2, 8, 3},
   {2, 0, 6, 5, 0, 4, 5, 2},
   {6, 1, 3, 6, 4, 5, 7, 4}}
  };


  if (channel < 2 && device < 5 && pin < 8) {
      result.dq_block = xio_block_lu[channel][device][pin];
      result.dq_pin = xio_pin_lu[channel][device][pin];
  }
  else {
      printf ("SYSLU_MBD: invalid value for channel, device, or pin\n");
      result.dq_block = 99;
      result.dq_pin = 99;
  }
  return result;
}
```

2.2.2.18 *Support Functions: SYSLU_PAT*

```
/* SYSLU_PAT*/
uint16 SYSLU_PAT (int pat_idx, struct XIO_pin pin) {
  return bit_pattern_dq[pat_idx * 36 + pin.dq_block * 9 + pin.dq_pin];
```

```
}
```

2.2.2.19 *Support Functions: SYSLU_PAT2*

```
/* SYSLU_PAT2: MC Address lookup given programmed device width and  pattern index */
uint32 SYSLU_PAT2 (int wprog, int pat_idx) {
  uint32 result;
  uint16 sc;

  /* Generates low-order bits of MIC address.  These bits control
     bank and col address bits.  High-order bits will be added
     by calling routine to map pattern to correct memory location.
     Input: pat_idx(5:0)
     Output: real address with selected row/col index

     +--------------------+--------+----------+------+----------+
     |    XX..XX          |Col(2:1)| Bank(2:0)|Col(0)| 0b000000 |
     +--------------------+--------+----------+------+----------+
     pat_idx:               5,4       3,2,1      0

     XDR                           Physical Address Bit
     Width     18  17  16  15  14  13  12  11 10 9  8  7  6  5  4  3  2  1  0
     ================================================================================
     x16       R2  R1  R0  C9  C8  C4  C7  C6 B2 B1 B0 Y  C5 x  x  x  x  x  x
     x8        R1  R0  C9  C8  C4  SC3 C7  C6 B2 B1 B0 Y  C5 x  x  x  x  x  x
     x4        R0  C9  C8  C4  SC3 SC2 C7  C6 B2 B1 B0 Y  C5 x  x  x  x  x  x

   */

  switch (wprog) {
    case 16:
      sc = 0;
      break;

    case 8:
      sc = (pat_idx & 0x2) >> 1;
      break;

    case 4:
      sc = (pat_idx & 0x6) >> 1;
      break;

    default:
      sc = 0;
  }

  result = (sc << 13) | ((pat_idx & 0x003E) << 7) | ((pat_idx & 0x0001) << 6);

  return result;
```

```
}
```

2.2.2.20 *Support Functions: WDSL_FMT*

```
/* WDSL_FMT: MC Data word to WDSL Format */
uint16 WDSL_FMT (uint16 mc_wd) {
  int n;
  uint16 result;

  uint16 bit_order[16] =
    { 15, 11, 7, 3, 14, 10, 6, 2, 13, 9, 5, 1, 12, 8, 4, 0 };

  /* First Generation x16/x8/x4 XDR DRAM with Burst Length == 16 */
  result = 0;
  for (n = 0; n < 16; n++) {
      if ((mc_wd >> bit_order[n]) & 0x0001)     {
          result |= (0x8000 >> n);
        }
  }
  return result;
}
```

2.2.2.21 *Support Functions: mic_cline_fmt*

```
/* mic_cline_fmt*/
int mic_cline_fmt (int n, uint16 * patt, struct cline_dq *cline) {
  int i, index;

  /* collect first 1/4 of cache line from low-order WDSL pattern bits */
  index = n * 2 * 36;
  for (i = 0; i < 4; i++)
    {
      cline[i].ecc = patt[index + 8] & 0x00FF;
      cline[i].data_hi =
        (patt[index + 7] & 0x00FF) << 24 | (patt[index + 6] & 0x00FF) << 16 |
        (patt[index + 5] & 0x00FF) << 8 | (patt[index + 4] & 0x00FF);
      cline[i].data_lo =
        (patt[index + 3] & 0x00FF) << 24 | (patt[index + 2] & 0x00FF) << 16 |
        (patt[index + 1] & 0x00FF) << 8 | (patt[index] & 0x00FF);
      index += 9;
    }
  /* collect second 1/4 of cache line from high-order WDSL pattern bits */
  index = n * 2 * 36;
  for (i = 4; i < 8; i++) {
      cline[i].ecc = (patt[index + 8] & 0xFF00) >> 8;
      cline[i].data_hi =
        (patt[index + 7] & 0xFF00) << 16 | (patt[index + 6] & 0xFF00) << 8 |
```

```
            (patt[index + 5] & 0xFF00) | (patt[index + 4] & 0xFF00) >> 8;
          cline[i].data_lo =
            (patt[index + 3] & 0xFF00) << 16 | (patt[index + 2] & 0xFF00) << 8 |
            (patt[index + 1] & 0xFF00) | (patt[index] & 0xFF00) >> 8;
          index += 9;
        }
      /* collect third 1/4 of cache line from low-order WDSL pattern bits */
      index = (n * 2 + 1) * 36;
      for (i = 8; i < 12; i++) {
          cline[i].ecc = patt[index + 8] & 0x00FF;
          cline[i].data_hi =
            (patt[index + 7] & 0x00FF) << 24 | (patt[index + 6] & 0x00FF) << 16 |
            (patt[index + 5] & 0x00FF) << 8 | (patt[index + 4] & 0x00FF);
          cline[i].data_lo =
            (patt[index + 3] & 0x00FF) << 24 | (patt[index + 2] & 0x00FF) << 16 |
            (patt[index + 1] & 0x00FF) << 8 | (patt[index] & 0x00FF);
          index += 9;
        }
      /* collect fourth 1/4 of cache line from high-order WDSL pattern bits */
      index = (n * 2 + 1) * 36;
      for (i = 12; i < 16; i++) {
          cline[i].ecc = (patt[index + 8] & 0xFF00) >> 8;
          cline[i].data_hi =
            (patt[index + 7] & 0xFF00) << 16 | (patt[index + 6] & 0xFF00) << 8 |
            (patt[index + 5] & 0xFF00) | (patt[index + 4] & 0xFF00) >> 8;
          cline[i].data_lo =
            (patt[index + 3] & 0xFF00) << 16 | (patt[index + 2] & 0xFF00) << 8 |
            (patt[index + 1] & 0xFF00) | (patt[index] & 0xFF00) >> 8;
          index += 9;
        }
       return 0;
    }
```

2.2.2.22 *Support Functions: mic_pattern_dq_load*

```
/* mic_pattern_dq_load */
void mic_pattern_dq_load (int lines, uint16 * patt) {

  int m, n, pat_index, addr_lsbs, address;

  mmio_write (MMIO_BE_MIC | YREG_INIT_CTL_0, 0x01400000, 0x00000000);
  delay_ns (100);
  mmio_write (MMIO_BE_MIC | YREG_INIT_CTL_1, 0x01400000, 0x00000000);
  delay_ns (100);

  for (m = 0; m < lines; m++) {
      mic_cline_fmt (m, patt, cline);

      pat_index = m * 2;
```

```
      addr_lsbs = SYSLU_PAT2 (XDR_WPROG, pat_index);
      address = WDSL_BASE_ADDR | addr_lsbs;

      mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_CUR_ADDR, 0x00000000, address);
      mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_ADDR, 0x00000000, 0x80000000);  /* select ECC */
      for (n = 0; n < 16; n++)  {
          mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_DATA, cline[n].ecc << 24,0x00000000);
      }

     mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_ADDR, 0x00000000, 0x00000000);  /* select data */
      for (n = 0; n < 16; n++)  {
          mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_DATA, cline[n].data_hi,
                      cline[n].data_lo);
      }

      mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_CUR_ADDR, 0x00000000, address + 0x80);
      mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_ADDR, 0x00000000, 0x80000000);  /* select ECC */
      for (n = 0; n < 16; n++)  {
          mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_DATA, cline[n].ecc << 24, 0x00000000);
      }

     mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_ADDR, 0x00000000, 0x00000000);  /* select data */
      for (n = 0; n < 16; n++)  {
          mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_DATA, cline[n].data_hi, cline[n].data_lo);
      }

    }
}
```

2.2.2.23 *Support Functions: XDR_store64*

```
/* XDR_store64 */
int XDR_store64 (uint32 address) {
  int n;
  uint32 init_val;

  /* Select low/high 64-byte sub-block */
  /* YREG_INIT_CTL(WDSLCLH) becomes least-significant column address bit */
  if (address & 0x0040) {
      init_val = 0x04010000;
  }
  else {
      init_val = 0x04000000;
  }

  mmio_write (MMIO_BE_MIC | YREG_INIT_CTL_0, init_val, 0x00000000); /* MIC pattern Mode */
  mmio_write (MMIO_BE_MIC | YREG_INIT_CTL_1, init_val, 0x00000000); /* MIC pattern Mode */

  address = (address & 0xFFFFFF80);
```

```
  /* Store address/data into AUX trace array. MIC will generate XDR DRAM store after 128 bytes */
  mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_CUR_ADDR, 0x00000000, address);
  mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_ADDR, 0x00000000, 0x00000000);      /* select data */

  for (n = 0; n < 16; n++) {
      mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_DATA, 0x00000000, 0x00000000);
  }
  return 0;
}
```

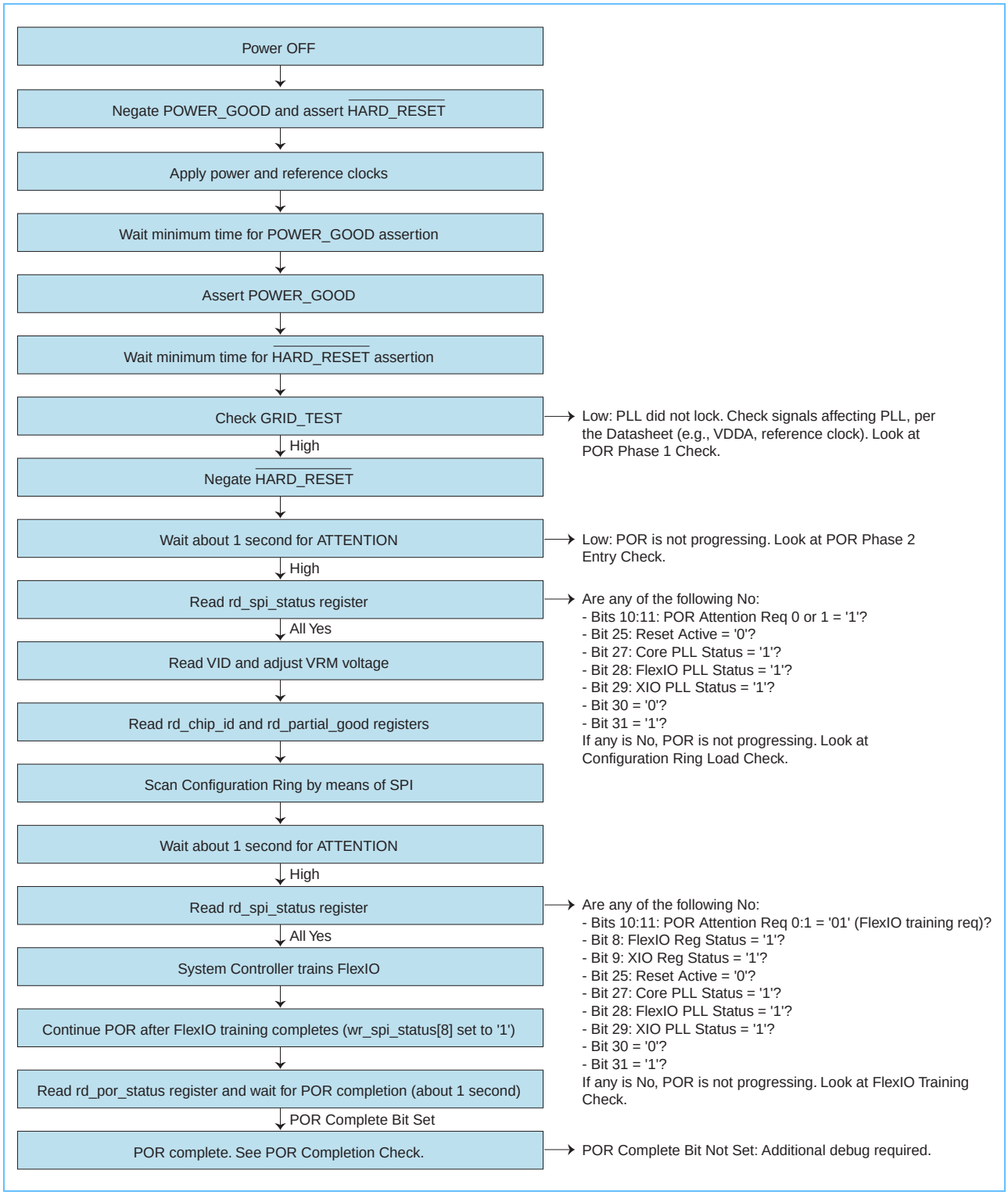
2.2.2.24 *Support Functions: XDR_store128*

```
/* XDR_store128 */
int XDR_store128 (uint32 address) {
  int n;

  address = (address & 0xFFFFFF80);

  /* Store address/data into AUX trace array. MIC will generate XDR DRAM store after 128 bytes */
  mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_CUR_ADDR, 0x00000000, address);

  mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_ADDR, 0x00000000, 0x00000000);      /* select data */
  for (n = 0; n < 16; n++) {
      mmio_write (MMIO_BE_MIC | MIC_AUX_TRC_GRF_DATA, 0x00000000, 0x00000000);
  }
  return 0;
}
```

2.3 Debug of the POR Sequence

Figure 2-8 on page 85 shows a debug process that can be used if the POR sequence of the CBE initialization is not completing as expected. The primary two SPI registers that can be used for monitoring the progress of the CBE POR state machine are the `rd_spi_status` and `rd_por_status` SPI registers.

Figure 2-8. POR Debug Flow



The sections below describe the phases of the POR sequence.

2.3.1 POR Phase 1 Check

POR Phase 1 is done almost completely internally to the CBE processor. Since the SPI interface is not available during POR Phase 1, the SPI status registers are also not available. However, the GRID_TEST signal can be used to verify that the Core PLL (clock source) is functioning correctly. This signal shows the lock status of the Core PLL, and should be active (locked) by the time the HARD_RESET signal can be inactive. If GRID_TEST does not become active, check the following:

- VDDA quality (voltage and filtering).
- Reference clock frequency (PLL_REFCLK) and signal leve.l
- Order in which power and reference clock are applied.

2.3.2 POR Phase 2 Entry Check

When the HARD_RESET signal goes inactive, the CBE processor will begin POR Phase 2. During this phase, there are two times at which intervention from the system controller is required. The first is the Configuration-Ring load, which should happen almost immediately after HARD_RESET is inactive. The System Controller should see the ATTENTION signal go active almost immediately after HARD_RESET is inactive.

If this does not happen, check the following:

- PLL locked (see *POR Phase 1 Check* on page 86).
- Signal level of the HARD_RESET signal, per recommendations in the *Cell Broadband Engine Datasheet*.
- Signal level of the ATTENTION signal, per the *Cell Broadband Engine Datasheet*.
- Signals that might affect POR:
  - CHECKSTOP_IN must be inactive.
  - PLL_CTL[0:1] must be tied inactive.
  - SPI_CTL[0:1] must match with the System Controller settings and follow recommendations in the *Cell Broadband Engine Datasheet*.
  - SYS_CONFIG[0:3] must be tied inactive.
  - TE must be tied inactive.
  - EXT_CLK_EN must be tied inactive.

2.3.3 RQ and DQ Debugging

For RQ debugging, there is no facility inside the MIC to aid with a command bus problem. Incorrect commands are simply not received by the XDR DRAMs correctly. Certain bits (such as address bits) may cause address parity errors.

For DQ debugging, (assuming a single bit error) the syndrome-to-pin mapping is shown in *Appendix D DQ Syndrome-to-Pin Mapping* on page 163.

2.3.4 Configuration-Ring Load Check

When the CBE processor drives the ATTENTION signal active, the reason for the Attention will be shown in the `rd_spi_status` register that is accessible from the SPI interface. At this point in POR Phase 2, the CBE processor should be requesting the Configuration-Ring data. Check the following by reading the `rd_spi_status` register:

- If the `rd_spi_status` register appears to be all 0's or all 1's, the data is not correctly read. Confirm that the Simple Read Sequence (*Section 3.3.2* on page 100) for reading this register matches what the CBE processor expects.
- `rd_spi_status[0]` reflects the ATTENTION signal. If this does not match the ATTENTION signal, confirm that the System Controller sequence matches with what the CBE processor expects.
- `rd_spi_status[1:4,7]` are Attention conditions caused by software (no software is involved at this point, so these should be all '0').
- `rd_spi_status[6]` is the thermal condition. This function is not enabled at this point, so this should be read as 0.
- `rd_spi_status[10:11]` shows the Attention request from the CBE POR state machine. These should appear as bit 10 = '1', bit 11 = '0'.
- `rd_spi_status[25]` is the inverse of the $\overline{\text{HARD\_RESET}}$ signal. This should always appear as '0' when the $\overline{\text{HARD\_RESET}}$ signal is inactive.
- `rd_spi_status[30:31]` are constants. These should appear as bit 30 = '0', bit 31 = '1'. If they do not, check the SPI sequence.

If the Attention was caused by the request for Configuration-Ring data, read the Voltage ID (VID) and adjust the Voltage Regulator Module (VRM) for VDD. See *Section 2.1.5.1 VRM adjustment with VID information* on page 34.

After the VRM is adjusted and the core VDD power supply has stabilized, read `rd_spi_status` again and confirm that bits 10:11 are still set to 1 and 0, respectively. The status should not have changed, because the CBE POR state machine should still be waiting for Configuration Ring to be loaded.

To confirm that reads of SPI registers are working correctly, the `rd_chip_id` register can be read. The values in this register are hardwired on the CBE processor for a specific version of the chip.

Before the Configuration-Ring data is loaded, the SPI `rd_partial_good` register must be read, because this data is needed to set the Configuration Ring with the correct SPE partial good information. See *Section 3.4.7 Read Partial Good Register (`rd_partial_good`)* on page 114. See *Section 4.2 Bit Descriptions* on page 120 for loading the Configuration Ring.

To summarize, check the following:

- Does the `rd_spi_status` register match the expected value?
- Was the VID adjusted correctly according to the *Cell Broadband Engine Datasheet*?
- Did the Configuration Ring get set with the partial good information from the `rd_partial_good` register?
- When scanning in the Configuration-Ring data, was the:
  - Address correct?

- SPI simple write sequence correct?
- Start bit appended to the data?
- Configuration data length matched the specified length for this CBE processor version?

2.3.5 FlexIO Calibration Check

After the CBE POR state machine acknowledges that the Configuration Ring has been scanned, it immediately progresses forward through the sequence and waits for the FlexIO calibration to be done. This step could be checked from the `rd_spi_status` register. Although this is done internally on the CBE processor, some of the operations done from CBE POR state machine affect the `rd_spi_status` register. These are:

- `rd_spi_status[8,9,28,29]` show the POR status of the FlexIO and XIO interfaces. These bits should all be '1' at this point in the sequence. If not, check the following:
  - The FlexIO and XIO interfaces run off separate PLLs from the core PLL. These PLLs are configured by the Configuration Ring. Confirm that the Configuration-Ring data for FlexIO and XIO all match the values recommended by Rambus for a specific version of the CBE processor.
  - Confirm that the FlexIO and XIO PLLs are set to the correct values and power supplies.

The CBE POR state machine waits for the System Controller to notify the CBE processor that the FlexIO calibration has completed. After the calibration is completed, the System Controller writes a '1' to the `wr_spi_status[8]` bit, and the CBE POR state machine finishes the POR sequence.

2.3.6 POR Sequence Completion Check

To determine whether the CBE POR state machine has completed the POR sequence, read the `rd_por_status` register and check for the following:

- `rd_por_status[0]` shows whether an error occurred during the POR sequence. This bit should be '0'.
- `rd_por_status[1]` shows whether any of the POR instructions did not complete as expected. This only applies to CBE internal instructions and not to external requests, such as a Configuration-Ring data request or a FlexIO calibration request. These external requests do not have any timer checking for duration.
- `rd_por_status[9]` shows whether FlexIO calibration completed. This bit should be '1'.
- `rd_por_status[11]` shows whether the CBE POR state machine has acknowledged the Configuration-Ring data. If this bit is '0', go back to *Section 2.3.4 Configuration-Ring Load Check* on page 87.
- `rd_por_status[17]` should be '1'.
- `rd_por_status[20]` should be '0'. If it is '1', check the SYS_CONFIG signal setting.
- `rd_por_status[22]` is the POR complete bit. This should be '1'.
- `rd_por_status[23]`should be '1'. If not, go back to *Section 2.3.2 POR Phase 2 Entry Check* on page 86.

2.3.7 Power-Off Sequence

See the *Cell Broadband Engine Datasheet* for minimum requirements for the power-off sequence.

3. Serial Peripheral Interface

The external configuration bus for the CBE processor is based on the Serial Peripheral Interface (SPI) specification. The SPI interface is a serial bus that has a master-slave relationship. The CBE processor is primarily targeted to participate on the SPI interface as a slave device. The external system controller accessing the SPI interface acts as the SPI Master.

The CBE processor's implementation of the SPI protocol differs from the SPI specification in only one respect—the End of SPI Cycle, described in *Section 3.1.1*.

3.1 SPI Operation

Table 3-1 shows the signals that are associated with the SPI Interface.

Table 3-1. SPI Signals

Name	Direction	Description
SPI_SI	input	Scan input data
SPI_SO	output	Scan output
SPI_CLK	input	SPI clock signal
SPI_EN	input	Enable signal

3.1.1 SPI Conventions

The following conventions are used:

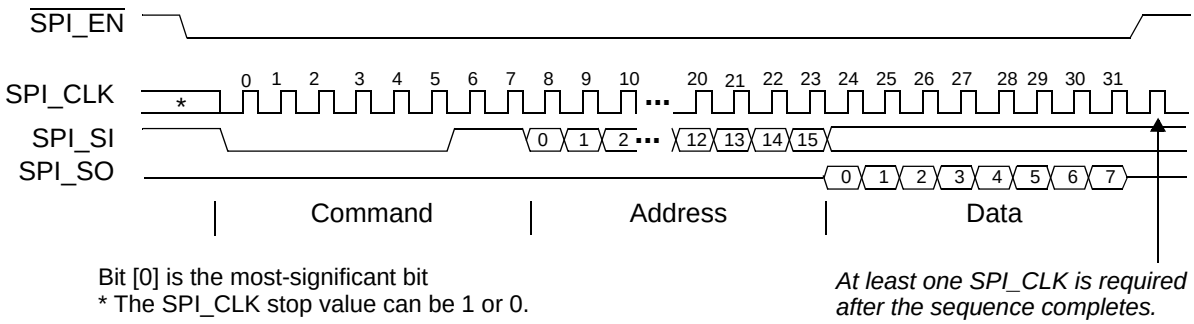
- *Bit Significance*—For serial data streams, the most-significant bit (bit [0]) must always be sent first for address, control, and data; only the `wr_config_ring` SPI register is an exception to this rule. The `wr_config_ring` register requires the least-significant bit to be scanned in first and the most-significant bit to be scanned in last.
- *Clocking*—The CBE processor samples serial input data (SPI_SI) on the rising edge of the serial clock and drives serial output data on the falling edge of the clock. The clock can be stopped during operation of the SPI interface. The recommended stop value for the clock is '0', although stopping the clock on '1' is also acceptable. The CBE processor uses edge-detection of the SPI clock to determine when to launch and capture data. The SPI clock does not drive internal state machines directly.
- *Start of SPI Cycle*—A valid SPI cycle is started on the first rising clock edge in which the SPI_EN enable signal is active. Activation of the enable signal is driven and removed on the falling edge of the clock.
- *End of SPI Cycle*—The removal of the enable signal signifies the end of a SPI cycle. The enable signal is removed on the falling edge of the clock. The CBE processor must receive at least one clock cycle with the enable signal deactivated between SPI transactions. The CBE processor takes no action while the enable signal is de-activated and the clock is running.

  Note: *The CBE-processor difference from a standard SPI protocol is that the CBE processor must receive at least one clock cycle with the enable signal deactivated between SPI transactions*.

3.2 SPI Protocol

The protocol for the SPI interface consists of a command, address, and data phase. *Figure 3-1* shows this protocol. The following sections provides additional detail on the protocol phases.

Figure 3-1. SPI Protocol



3.2.1 SPI Command

The SPI command is eight bits long. The command is defined as the first eight bits after the enable signal goes low. The command is sub-divided into three fields: CBE Chip ID, Multi-chip ID, and Command. *Figure 3-2* shows the command format, and *Table 3-2* on page 93 shows the definition of command bits.

Note: *The multi-chip ID needs to match the value on the SPI_CTL[0:1] pins.*

Figure 3-2. SPI Command Format

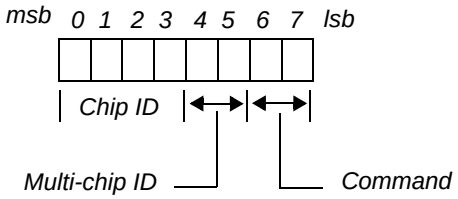


Table 3-2. SPI Command Bit Definition

Bits	Function	Bit Definition	Description
0:3	CBE Chip ID	0000	Serial SPI Memory
		0001	CBE processor
		0010	IOIF0 device
		0011	IOIF1 device
		0100	System Controller
		011x	Reserved
		1xxx	Reserved
4:5	Multi-chip ID	00	CBE processor 0
		01	CBE processor 1
		10	CBE processor 2
		01	CBE processor 3
6:7	Command	00	Read Command
		01	Write Command
		1x	Reserved

3.2.2 SPI Address

An SPI address is always 16 bits. *Table 3-3* on page 94 shows the address assignments for the different blocks of the CBE processor. *Table 3-4* on page 94 defines the mapping of SPI addresses to MMIO registers. Additional tables, beginning with *Table 3-5* on page 94, define which MMIO registers can be accessed through the SPI interface. SPI registers are described in *Section 3.4* on page 107. For bit definitions and control information for FlexIO registers, see *Rambus FlexIO Processor Bus Interface Cell Datasheet (DL-0159)* and *Rambus BE-FlexIO Processor Bus Interface Cell - Addendum to rev 0.90 FlexIO Processor Bus Interface Cell Datasheet (DL-0159)*.

The SPI access sequences are different, depending on the address range that is accessed. There are two types of SPI registers within the Pervasive Logic:

- SPI registers that are accessed through simple reads and writes.
- MMIO registers and FlexIO registers that are accessed through Internal Configuration Bus (ICB) reads and writes.

Each sequence is described separately in the following sections. *Section 3.3.4* on page 101 gives an overview of the ICB.

Table 3-3. SPI Address Map

SPI Address Range		CBE Block Addressed	Access Sequence	Details
Lower	Upper			
x'0000'	x'1000'	SPI Registers in Pervasive Logic	Simple reads and writes	SPI registers defined in this section are accessed only by means of the SPI Interface. (For this reason, the are not are described in the *Cell Broadband Engine Registers* document.)
x'2000'	x'8FFF'	Reserved		
x'9000'	x'9FFF'	MMIO Registers in Pervasive Logic	Internal Configuration Bus (ICB)	These MMIO registers are defined in the *Cell Broadband Engine Registers* document, but can be accessed indirectly through SPI ICB sequences.
x'A000'	x'AFFF'	Reserved		
x'B000'	x'CFFF'	Reserved		
x'D000'	x'FFFF'	Cell Broadband Engine Interface (BEI)		These MMIO registers are defined in the *Cell Broadband Engine Registers* document, but can be accessed indirectly through SPI ICB sequences.

Table 3-4. SPI-Address Mapping to MMIO Registers Through the ICB

SPI Address Range		ICB Range		MMIO Register Range		Description
Lower	Upper	Lower	Upper	Lower	Upper	
x'9000'	x'9FFF'	x'1000'	x'1FFF'	x'509000'	x'509FFF'	MMIO Registers in Pervasive Logic
x'A000'	x'AFFF'	x'2000'	x'2FFF'			Reserved
x'B000'	x'CFFF'	x'3000'	x'4FFF'			Reserved
x'D000'	x'D3FF'	x'5000'	x'53FF'	x'511000'	x'5113FF'	BEI BIC 0 (NClk)[1]
x'D400'	x'D7FF'	x'5400'	x'57FF'	x'511400'	x'5117FF'	BEI BIC 1 (NClk)[1]
x'D800'	x'DBFF'	x'5800'	x'5BFF'	x'511800'	x'511BFF'	BEI EIB
x'DC00'	x'DFFF'	x'6C00'	x'6FFF'	x'511C00'	x'511FFF'	BEI IOC I/O Command
x'E000'	x'EFFF'	x'6000'	x'6FFF'	x'512000'	x'512FFF'	BEI BIC 0 (BClk)[1]
x'F000'	x'FFFF'	x'7000'	x'7FFF'	x'513000'	x'513FFF'	BEI BIC 1 (BClk)[1]

1. The BIC is the Bus Interface Controller block of the Broadband Engine Interface (BEI) to I/O.

Table 3-5. SPI Registers in Pervasive Logic (Page 1 of 2)

SPI Address	MMIO Register Name	Width (Bits)	Read/ Write	Additional Information
x'0000'	Read SPI Status (rd_spI_status)	32	R	*SPI Status Register* on page 107.
	Write SPI Status (wr_spi_status)	32	W	*SPI Status Register* on page 107.
x'0001'	Write Configuration Ring (wr_config_ring)	2697	W	*Write Configuration Ring (`wr_config_ring`)* on page 110.

Table 3-5. SPI Registers in Pervasive Logic (Page 2 of 2)

SPI Address	MMIO Register Name	Width (Bits)	Read/ Write	Additional Information
x'0002'	ICB Poll (icb_poll)	8	R	*ICB Poll Register (icb_poll)* on page 110.
x'0003'	Read CBE Chip ID (rd_chip_id)	32	R	*Read CBE Chip ID (rd_chip_id)* on page 111.
x'0004'	Reserved	32	R	
x'0005'	Reserved	32	R	
x'0006'	Reserved	32	R	
x'0007'	Reserved	32	R	
x'000A'	Read Serial Number Bit 0:31 (rd_serial_num0)	32	R	*Read Serial Number Register(rd_serial_num0, rd_serial_num1)* on page 112.
x'000B'	Read Serial Number Bit 32:47 (rd_serial_num1)	32	R	
x'000C'	Read Voltage ID (rd_VID)	8	R	*Read Voltage ID (rd_VID)* on page 113.
x'000D'	Read Partial Good Information (rd_partial_good)	8	R	*Read Partial Good Register (rd_partial_good)* on page 114.
x'000E'	Read Linear Thermal Diode Calibration Register (rd_lin_therm_diode)	24	R	*Read Linear Thermal Diode Calibration Register (rd_lin_therm_diode)* on page 115.
x'000F'	Read POR status (rd_por_status)	32	R	*Read POR Status Register (rd_por_status)* on page 116.
x'0010'	Read ICB Data (rd_icb_data)	64	R	*Read ICB Data Register (rd_icb_data)* on page 117.
x'0020'	Reserved	65	W	

Table 3-4 through *Table 3-9* summarize the MMIO registers in the Pervasive Logic and their SPI addresses. Bit definitions for these registers are given in the *Cell Broadband Engine Registers* document.

Table 3-6. MMIO Registers in Pervasive Logic (Page 1 of 3)

SPI Address	MMIO Register Name	Width (Bits)	Read/ Write
Reliability, Availability, and Serviceability (RAS) Registers[1]			
x'9C00'	*Global Fault Isolation Register (checkstop_fir)*	32	R/W
x'9C08'	*Global Fault Isolation Register For Recoverable Errors (recoverable_fir)*	32	R/W
x'9C10'	*Global Fault Isolation Register For Special Attention And Machine Check (spec_att_mchk_fir)*	32	R/W
x'9C18'	*Global Fault Isolation Mode Register (fir_mode_reg)*	32	R/W
x'9C20'	*Global Fault Isolation Error Enable Mask Register (fir_enable_mask)*	32	R/W
x'9C28'	*Local Error Counter Status Register (loc_cn_status_reg)*	32	R/W
x'9C30'	Reserved		
x'9C38'	*SPE Available Partial Good Register (SPE_available)*	32	R
x'9C40'	*Hold Request Register (hold_request)*	64	R/W

Table 3-6. MMIO Registers in Pervasive Logic (Page 2 of 3)

SPI Address	MMIO Register Name	Width (Bits)	Read/ Write
x‘9C48’	*Reserved*		
x‘9C50’	*Reserved*		
x‘9C58’	*Reserved*		
x‘9C80’	*Serial Number (serial_number)*	64	R
x‘9C88’	Reserved		
x‘9C90’	Reserved		
x‘9C98’	Reserved		
x‘9CA0’	Reserved		
x‘9CA8’	Reserved		
x‘9CB0’	Reserved		
x‘9CB8’	Reserved		
Performance Monitor Registers			
x‘9008’	*Group Control Register (group_control)*	32	W
x‘90A8’	*Debug Bus Control Register (debug_bus_control)*	32	W
x‘9108’	*Trace Buffer High Doubleword Register (0 to 63) (trace_buffer_high)*	64	R
x‘9110’	*Trace Buffer Low Doubleword Register (64 to 127) (trace_buffer_low)*	64	R
x‘9118’	*Trace Address Register (trace_address)*	32	R/W
x‘9120’	*External Trace Timer Register (ext_tr_timer)*	64	W
x‘9400’	*Performance Monitor Status/Interrupt Mask Register (pm_status)*	32	R/W
x‘9408’	*Performance Monitor Control Register (pm_control)*	32	W
x‘9410’	*Performance Monitor Interval Register (pm_interval)*	32	R/W
x‘9418’	*Performance Monitor Counter Pairs Register (pmM_N)*	32	R/W
x‘9420’			
x‘9428’			
x‘9430’			
x‘9438’	*PM Start Stop (pm_start_stop)*	32	W
x‘9440’	*Performance Monitor Counter Control Registers (pmN_control)*	16	W
x‘9448’			
x‘9450’			
x‘9458’			
x‘9460’			
x‘9468’			
x‘9470’			
x‘9478’			

Table 3-6. MMIO Registers in Pervasive Logic (Page 3 of 3)

SPI Address	MMIO Register Name	Width (Bits)	Read/ Write
Power Management Control Registers (Accessed through ICB read/write)			
x‘9880’	*Power Management Control Register (PMCR)*	64	R/W
x‘9888’	*Power Management Status Register (PMSR)*	64	R
Thermal Management MMIO Registers (Accessed through ICB read/write)			
x‘9800’	*Thermal Sensor Current Temperature Status Register 1 (TS_CTSR1)*	64	R
x‘9808’	*Thermal Sensor Current Temperature Status Register 2 (TS_CTSR2)*	64	R
x‘9810’	*Thermal Sensor Maximum Temperature Status Register 1 (TS_MTSR1)*	64	R
x‘9818’	*Thermal Sensor Maximum Temperature Status Register 2 (TS_MTSR2)*	64	R
x‘9820’	*Thermal Sensor Interrupt Temperature Register 1 (TS_ITR1)*	64	R/W
x‘9828’	*Thermal Sensor Interrupt Temperature Register 2 (TS_ITR2)*	64	R/W
x‘9830’	*Thermal Sensor Global Interrupt Temperature Register (TS_GITR)*	64	R/W
x‘9838’	*Thermal Sensor Interrupt Status Register (TS_ISR)*	64	R/W
x‘9840’	*Thermal Sensor Interrupt Mask Register (TS_IMR)*	64	R/W
x‘9848’	*Thermal Management Control Register 1 (TM_CR1)*	64	R/W
x‘9850’	*Thermal Management Control Register 2 (TM_CR2)*	64	R/W
x‘9858’	*Thermal Management System Interrupt Mask Register (TM_SIMR)*	64	R/W
x‘9860’	*Thermal Management Throttle Point Register (TM_TPR)*	64	R/W
x‘9868’	*Thermal Management Stop Time Register 1 (TM_STR1)*	64	R/W
x‘9870’	*Thermal Management Stop Time Register 2 (TM_STR2)*	64	R/W
x‘9878’	*Thermal Management Throttle Scale Register (TM_TSR)*	64	R/W
Time Base Registers (Accessed through ICB read/write)			
x‘9890’	*Time Base Register (TBR)*	64	R/W

1. Also called Test Control Unit (TCU).

Table 3-7. BEI EIB (Page 1 of 2)

SPI Address	MMIO Register Name	Width (Bits)	Read/ Write
x‘D800’	*EIB AC0 Control Register (EIB_AC0_CTL)*	64	R/W
x‘D808’	Reserved		
x‘D810’	*EIB Interrupt Register (EIB_Int)*	64	R/W
x‘D840’	*EIB Local Base Address Register 0 (EIB_LBAR0)*	64	R/W
x‘D848’	*EIB Local Base Address Mask Register 0 (EIB_LBAMR0)*	64	R/W
x‘D850’	*EIB Local Base Address Register 1 (EIB_LBAR1)*	64	R/W
x‘D858’	*EIB Local Base Address Mask Register 1 (EIB_LBAMR1)*	64	R/W
x‘D860’	Reserved		
x‘D868’	Reserved		

Table 3-7. BEI EIB (Page 2 of 2)

SPI Address	MMIO Register Name	Width (Bits)	Read/ Write
x‘D870’	*EIB AC/Darb Configuration Register (EIB_Cfg)*	64	R/W
x‘D878’	*EIB Address Concentrator Overrun Isolation Register (EIB_AC_Ovr)*	64	R/W
x‘D880’ – x‘DBFF’	Reserved		

Table 3-8. BEI IOC Command

SPI Address	MMIO Register Name	Width (Bits)	Read/ Write
x‘DC00’	*IOCmd Configuration Register (IOC_IOCmd_Cfg)*	64	R/W
x‘DC08’	*IOC Memory Base Address Register (IOC_MemBaseAddr)*	64	R/W
x‘DC10’	*IOC Base Address Register 0 (IOC_BaseAddr0)*	64	R/W
x‘DC18’	*IOC Base Address Mask Register 0 (IOC_BaseAddrMask0)*	64	R/W
x‘DC20’	*IOC Base Address Register 1 (IOC_BaseAddr1)*	64	R/W
x‘DC28’	*IOC Base Address Mask Register 1 (IOC_BaseAddrMask1)*	64	R/W
x‘DC30’	Reserved		
x‘DC38’	Reserved		
x‘DC40’	Reserved		
x‘DC48’	Reserved		
x‘DC50’	Reserved		
x‘DC58’	*IOC SRAM Parity Error Capture Register (IOC_SRAM_ParityErrCap)*	64	R
x‘DC60’ – x‘DFFF’	Reserved		

Table 3-9. BEI BIC 0/1 on the BClk

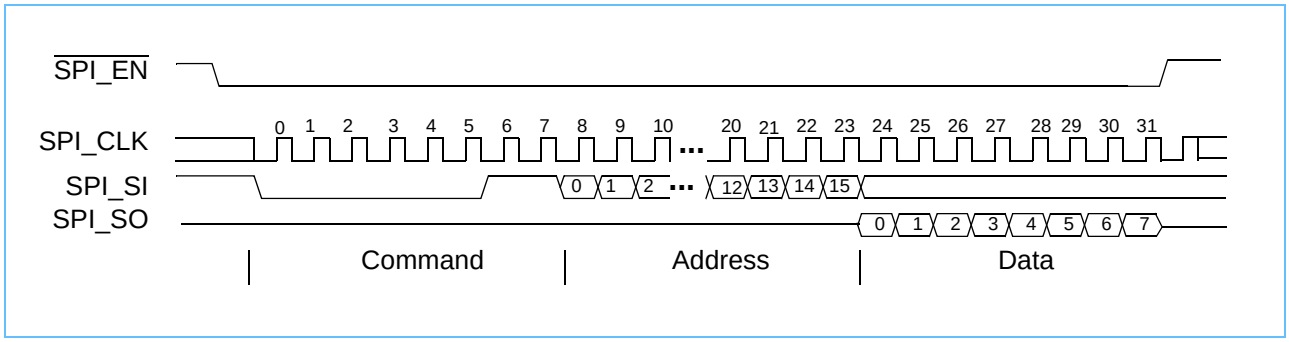
SPI Address	MMIO Register Name	Width (Bits)	Read/ Write
x‘F600’ – Link 0 x‘F608’ – Link 1	*BED Link n [n = 0, 1] Transmit Byte Training Control Registers (BED_Lnk0_TransBytTrngCntl, BED_Lnk1_TransBytTrngCntl)*	32	R/W
x‘F610’ – Link 0 x‘F618’ – Link 1	*BED Link n [n = 0, 1] Receive Byte Training Control Registers (BED_RecBytTrngCntl_Lnk0, BED_RecBytTrngCntl_Lnk1)*	32	R/W
x‘F620’	*BED RRAC Register Control Register (BED_RRAC_RegCntl)*	32	R/W
x‘F628’	*BED RRAC Register Read Data Register (BED_RRAC_RegRdDat)*	32	R/W
x‘F630’	Reserved		

3.2.3 SPI Data

The SPI data length can be variable. Deactivation of the enable signal signifies the end of a data transfer. This mechanism allows the transmission of a single bit or a large number of bits. *Figure 3-3* shows the timing sequence for transferring a single byte of data. If additional data is required for the transfer shown, the enable signal simply stays low for the required number of clocks.

Data is transmitted most-significant bit first; only the `wr_config_ring` register is an exception to this rule. The `wr_config_ring` register requires the least-significant bit to be scanned in first and the most-significant bit to be scanned in last.

Figure 3-3. SPI Data Byte Transfer



3.3 SPI Sequence Types

The SPI interface supports seven sequence types:

- Simple write to SPI register.
- Simple read from SPI register.
- Poll (implemented as simple read) SPI register.
- ICB write to MMIO register.
- ICB read from MMIO register.
- ICB indirect write to FlexIO register.
- ICB indirect read to FlexIO register.

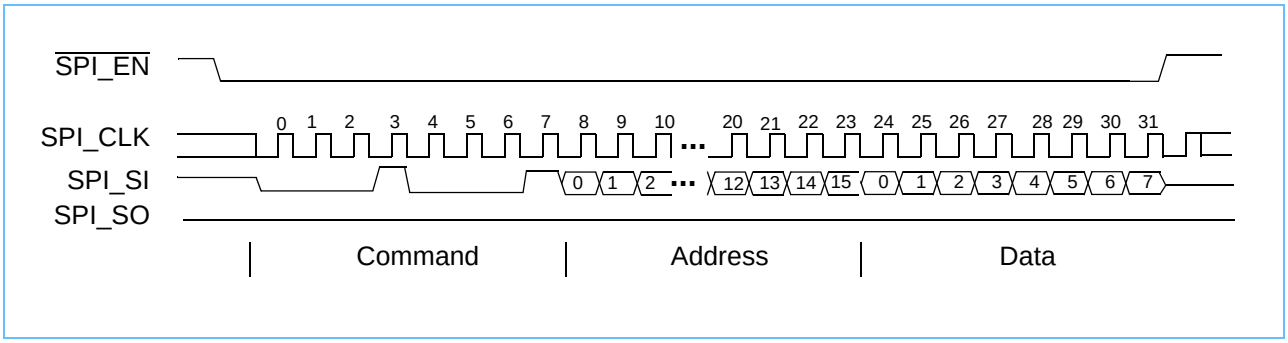
These are referred to as *SPI registers* in CBE documentation, and they are described in *Section 3.4* on page 107. SPI registers are defined only in this document, not in the *Cell Broadband Engine Registers* document, because SPI registers are not software accessible.

The *Cell Broadband Engine Registers* document defines MMIO and SPR software-programmable registers. Pervasive-Logic and BEI MMIO registers can be accessed using the ICB sequence types—ICB write or ICB read—whereas internal Rambus FlexIO registers are accessed as ICB indirect-read and indirect-write sequences.

3.3.1 Simple Write Sequence

A simple write sequence is defined as a sequence with SPI command bit [7] set to ‘1’. The transaction shown in *Figure 3-4* is a write to the CBE chip ID as CBE chip 0. Bit [3] indicates that the sequence is targeted for a CBE processor. The data portion of the write sequence is variable and depends on the number of bits in the SPI register.

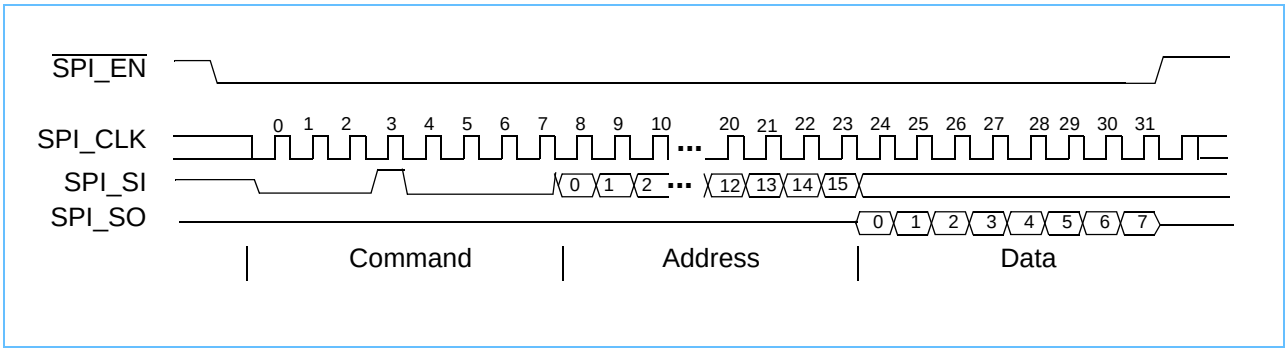
Figure 3-4. SPI Simple Write Sequence



3.3.2 Simple Read Sequence

A simple read sequence is defined as command bit [7] set as ‘0’. The data will appear on the clock cycle following the 16-bit address. *Figure 3-5* shows a read transaction. The data portion of the read sequence is variable and depends on the number of bits in the SPI register; the simple read sequence is designed to accommodate this variable length.

Figure 3-5. SPI Simple Read Sequence



In a system that can only read 64 bits at a time, the data read from registers smaller than 64 bits is wrapped around and occupies the extra bits also. For example, a 64-bit read of a 32-bit read register results in the 32 bits being duplicated in bits 0:31 and 32:63. See *Section 3.4.10* on page 117 for additional details relating to the Read ICB Data Register (`rd_icb_data`).

3.3.3 Polling

Polling is a simple read sequence of the 8-bit `icb_poll` register at SPI address x’0002’. The `icb_poll` register is used as part of the ICB read and ICB indirect-read sequences. See the bit definitions for the register in *Section 3.4.3* on page 110.

The poll transaction looks exactly the same as a simple read operation. The poll is done after an ICB read command. The `icb_poll` register returns a single byte of valid data. If the ICB read data is not ready, the Pervasive Logic sends back all zeros in the data (x'00'). If the read data is ready, the Pervasive Logic responds with a non-zero data value (x'80').

Poll transactions can be read any time the SPI interface is available. The System Controller must implement a time-out counter to insure that the transaction completes in a timely fashion. If the time-out counter expires and the `icb_poll` register still indicates that the data is not ready, the System Controller must issue a Clear ICB command to abort the operation and clear the internal state machine of the Pervasive Logic. The Clear ICB command is accomplished by writing bit [0] of the `wr_spi_status` register to '1'.

The minimum value of the time-out counter depends on the speed of the SPI, but (assuming an SPI master that reads the SPI at a clock rate as slow as 100MHz) it should represent no more than three attempts to read the `icb_poll` register. With a substantially a higher clock rate, only one attempt to read the `icb_poll` register should normally be needed. The maximum latency for a read of the `icb_poll` register will be 160 `NClk`/2 cycles, or 320 `NClk` cycles.

3.3.4 ICB Sequences

3.3.4.1 *ICB Communication with MMIO Registers*

The Internal Configuration Bus (ICB) is an internal serial interface used to communicate with on-chip devices. The ICB communicates with the internal MMIO bus to access registers available on it. The Pervasive Logic is responsible for receiving and sending information to and from the SPI interface. This logic translates the SPI information into a format used by the ICB. The ICB logic operates at half the CBE core clock (`NClk`/2).

Registers are not directly accessible to ICB sequences from the SPI interface. ICB sequences rely on the Pervasive Logic to translate requests into MMIO-bus accesses. For ICB sequences—including the indirect types—the SPI interface will issue a request to the Pervasive Logic when access to these registers is requested. This translation can only happen when the MMIO bus is available for the SPI interface to use.

On a read operation to an MMIO register through the ICB, the poll sequence must be used to monitor that the request has really been placed on the MMIO bus and that a reply (read data) has been returned and is ready. No such poll sequence is required for a write operation to an MMIO register through the ICB, because the request will eventually be placed upon the MMIO bus. No acknowledgement is sent back to the System Controller, so no polling sequence is required.

All accesses to the ICB interface—reads and writes—appear as simple SPI sequences with 64 bits of data (8-bit command, 16-bit address, 64-bit data). Therefore, all transfers to the ICB interface are a total of 88-bits.

3.3.4.2 *ICB Write Example*

The following example shows how to write the Performance Monitor (`pm_interval`) register on the ICB interface. The variables required to write the Performance Monitor (`pm_interval`) register are:

- SPI Command = x'11'.

- SPI Address = x‘9410’.
- SPI Data = x‘12345678_00000000’.

These 88 bits are written on the SPI interface to cause a write to occur to the Performance Monitor. *Table 3-10* shows the bit stream required for the SPI transaction to complete the above sequence.

Table 3-10. Example SPI Bit Stream for ICB Write

SPI Bits	Function	Contents	Description/ Comment
0:7	Command	x‘11’	This command selects CBE chip ID = ‘0001’, multi-chip ID = ‘00’, and write command = ‘01’.
8:23	Address	x‘9410’	16-bit address that points to pm_interval register on the ICB interface.
24:55	Data	x‘12345678’	Write pattern data to PM.
56:87	Data	x‘00000000’	Fill data needed to complete the 88-bit SPI transaction.

3.3.4.3 *ICB Read Example*

A read of an ICB device by means of the SPI interface requires a minimum of three SPI transactions, as follows:

1. Send an 88-bit ICB read command to the desired address. The data sent as part of this transaction is dummy data.
2. Send a 32-bit read command to the ICB Poll (`icb_poll`) register. Loop here until the `icb_poll` register is non-zero. (See *Section 3.4.3* on page 110 for details on the `icb_poll` register.)
3. Send a simple read command to the `rd_icb_data` SPI register to get the ICB data. (See *Section 3.4.10* on page 117 for details on the `rd_icb_data` register.)

The following example shows how to read the `trace_buffer_high` MMIO register:

1. Send ICB read request (64-bit data read):
   - SPI Command = x‘10’.
   - SPI Address = x‘9108’.
   - SPI Data = x‘00000000_00000000’ (dummy data).
2. Read status of read operation (8-bit data read):
   - SPI Command = x‘10’.
   - SPI Address = x‘0002’.
   - SPI Data = 8 bits of status returned from the CBE processor. `icb_poll[0]` is ‘1’ when the read data is ready.
3. Read return data (64-bit read):
   - SPI Command = x‘10’.
   - SPI Address = x‘0010’.
   - SPI Data = 64 bits of data returned from Performance Monitor.

Table 3-11 shows the three SPI bit streams that are required to perform the sequence described above.

Table 3-11. Example SPI Bit Stream to Read the Performance Monitor Trace Buffer

SPI Bits	Function	Contents	Description
Send Read Request to ICB (SPI Transaction 1)			
0:7	Command	x‘10’	This command selects CBE chip ID = ‘0001’, multi-chip ID = ‘00’, and read command = ‘00’.
8:23	Address	x‘9108’	16-bit address that points to trace_buffer_high register on the ICB interface.
24:55	Data	x‘0000’	Fill data needed to complete the 88-bit SPI transaction.
56:87	Data	x‘00000000’	Fill data needed to complete the 88-bit SPI transaction.
Read icb_poll register from Pervasive Logic (SPI Transaction 2)			
0:7	Command	x‘10’	This command selects CBE chip ID = ‘0001’, multi-chip ID = ‘00’, and read command = ‘00’.
8:23	Address	x‘0002’	16-bit address that points to the icb_poll register.
24:31	Data	x‘80’	Most-significant bit indicates read data is ready.
Read ICB Data (PM trace buffer(0-63) from Pervasive Logic (SPI Transaction 3)			
0:7	Command	x‘10’	This command selects CBE chip ID = ‘0001’, multi-chip ID = ‘00’, and read command = ‘00’.
8:23	Address	x‘0010’	16-bit address that points to the rd_icb_data register.
24:87	Data	x‘xxxxxxxx_ xxxxxxxx‘	Read 64 bits of data from PM trace buffer 0-63.

3.3.4.4 *ICB Indirect Access to FlexIO*

Rambus FlexIO registers internal to the CBE processor are indirectly accessed through Bus Interface Controller (BIC) MMIO registers, `BED_RRAC_RegCntl` (for FlexIO control and write data) and `BED_RRAC_RegRdDat` (for FlexIO read data). Access is gained through the FlexIO Register Interface facility in the BIC, which is the proxy for the FlexIO registers mapped onto the MMIO bus. See the *Cell Broadband Engine Registers* document for bit definitions of these MMIO registers.

Table 3-12 shows the SPI addresses used for transactions to the FlexIO register space. The read and write data on the SPI interface is 64 bits (as part of the 88 bit SPI sequence), with some of the data padded with zeros because the MMIO registers are 32 bit registers and the data width of the Rambus FlexIO registers is 16 bits.

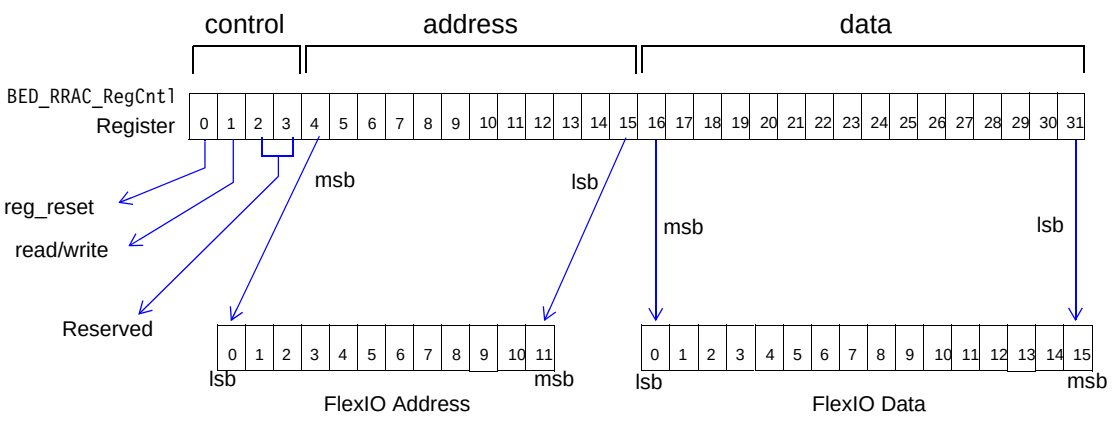
For bit definitions and control information for FlexIO registers, see *Rambus FlexIO Processor Bus Interface Cell Datasheet (DL-0159)* and *Rambus BE-FlexIO Processor Bus Interface Cell - Addendum to rev 0.90 FlexIO Processor Bus Interface Cell Datasheet (DL-0159)*.

Table 3-12. SPI FlexIO Related Addresses

SPI Address	Data Width	Description
x’F620’	64	BIC FlexIO R/W control register (BED_RRAC_RegCntl MMIO register).
x’F628’	64	BIC FlexIO read data register (BED_RRAC_RegRdDat MMIO register).
x’0002’	8	Pervasive-Logic read status register (icb_poll).
x’0010’	16	Pervasive-Logic read data register (rd_icb_data).

Figure 3-6 shows the mapping of the BED_RRAC_RegCntl MMIO register to FlexIO Address and FlexIO Data. The bit ordering is reversed on the Rambus interface.

Figure 3-6. BED_RRAC_RegCntl MMIO register *mapping to FlexIO Address and FlexIO Data*



3.3.4.5 *ICB Indirect Write to FlexIO Example*

The MMIO register, BED_RRAC_RegCntl, is used to write to all of the FlexIO registers on the CBE processor. The following example shows an ICB indirect write to the Rambus FlexIO register, BX_CTL, which turns on the BX_CTL block enable bit:

- SPI Control = x‘11’.
- SPI Address = x‘F620’.
- SPI Data = x‘400C0001_00000000’.

The above 88 bits are written on the SPI interface to cause a write to occur to the BX_CTL register by means of the BED_RRAC_RegCntl MMIO register. *Table 3-13* shows the bit stream require for the SPI transaction to complete the above sequence.

Table 3-13. Example SPI Bit Stream to Write FlexIO BX_CTL Reg

SPI Bits	Function	Contents	Description/ Comment
0:7	Command	x‘11’	This command selects CBE chip ID = ‘0001’, multi-chip ID = ‘00’,and write command = 01.
8:23	Address	x‘F620’	16-bit address that points to the BIC FlexIO Control register (BED_RRAC_RegCntl).
24:27	Data	x‘4’	FlexIO register control field (BED_RRAC_RegCntl MMIO[0:3]).
28-39	Data	x‘00C’	Select block ‘0000’, pin select = ‘0000’, the BX_CTL register = ‘1100’. (BED_RRAC_RegCntl MMIO[4:15]).
40:55	Data	x‘8000’	Turn on block enable bit (FlexIO data bit [0]). (BED_RRAC_RegCntl MMIO[16:31]).
56:87	Data	x‘00000000’	Fill data needed to complete the 88-bit SPI transaction.

3.3.4.6 *ICB Indirect Read to FlexIO Example*

ICB indirect reads to the FlexIO registers require a minimum of four SPI transactions:

1. Send read address command to the BIC.
2. Send read data request to BIC.
3. Read status of BIC operation from Pervasive Logic. Repeat this operation until the read ready indicator is active through the `icb_poll` register.
4. Read the 16-bit FlexIO return data from Pervasive Logic.

The SPI transactions to read the Rambus FlexIO `RRAC_ID` register consists of the following sequences:

1. Send the FlexIO register address to the BIC (64-bit data write):
   - SPI Command = x‘11’.
   - SPI Address = x‘F620’.
   - SPI Data = x‘00000000_00000000’.
2. Send the FlexIO Read Data Request command to the BIC (64-bit data read):
   - SPI Command = x‘10’.
   - SPI Address = x‘F628’.
   - SPI Data = x‘00000000_00000000’ (dummy data).
3. Read the status of read operation (8-bit data read) from the `icb_poll` registers:
   - SPI Command = x‘10’.
   - SPI Address = x‘002’.
   - SPI Data = 8-bit status returned from CBE processor. Data bit [0] is ‘1’ when read data is ready.
4. Read the ICB return data (64-bit read) from the `rd_icb_data` register:
   - SPI Command = x‘10’.
   - SPI Address = x‘0010’.
   - SPI Data = 16-bits of data returned from the FlexIO register by means of the BIC. The remaining 48 bits are zero.

Table 3-14 shows the three SPI bit streams required to perform the sequence described above.

Table 3-14. Example SPI Bit Stream to Read FlexIO RRAC_ID Register (Page 1 of 2)

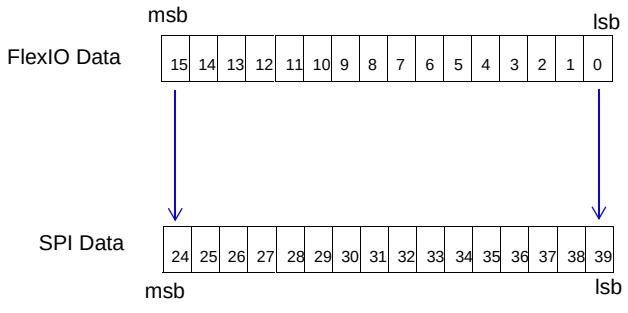
SPI Bits	Function	Contents	Description
Send Read Address Command to BIC (SPI Transaction 1)			
0:7	Command	x‘11’	This command selects CBE chip ID = ‘0001’, multi-chip ID = ‘00’,and write command = 01.
8:23	Address	x‘F620’	16-bit address that points to the BIC FlexIO Control register (BED_RRAC_RegCntl).
24:27	Data	x‘0’	FlexIO register control field (BED_RRAC_RegCntl MMIO[0:3]) which selects a FlexIO register read.

Table 3-14. Example SPI Bit Stream to Read FlexIO RRAC_ID Register (Page 2 of 2)

SPI Bits	Function	Contents	Description
28-39	Data	x‘000’	Select block ‘0000’, pin select = ‘0000’, the RRAC_ID register = ‘0000’. (BED_RRAC_RegCntl MMIO[4:15]).
40:87	Data	x‘0000_‘00000000’	Dummy Fill data needed to complete the 88-bit SPI transaction.
Send Read Data Request Command to BIC (SPI Transaction 2)			
0:7	Command	x‘10’	This command selects CBE chip ID = ‘0001’, multi-chip ID = ‘00’, and read command = ‘00’.
8:23	Address	x‘F628’	16-bit address that points to the BIC FlexIO read data register (BED_RRAC_RegRdDat MMIO register).
24:27	Data	x‘0’	BIC FlexIO Control register control field.
28-39	Data	x‘000’	Fill data needed to complete the 88-bit SPI transaction.
40:55	Data	x‘0000’	Fill data needed to complete the 88-bit SPI transaction.
56:87	Data	x‘00000000’	Fill data needed to complete the 88-bit SPI transaction.
Read ICB Poll status register from Pervasive Logic (SPI Transaction 3)			
0:7	Command	x‘10’	This command selects CBE chip ID = ‘0001’, multi-chip ID = ‘00’, and read command = ‘00’.
8:23	Address	x‘0002’	16-bit address that points to the icb_poll register.
24:31	Data	’ x’80’	Most-significant bit indicates read data is ready.
Read ICB Data (RRAC_ID) from Pervasive Logic by means of BIC/FlexIO (SPI Transaction 4)			
0:7	Command	x‘10’	This command selects CBE chip ID = ‘0001’, multi-chip ID = ‘00’, and read command = ‘00’.
8:23	Address	x‘0010’	16-bit address that points to the rd_icb_data register.
24:39	Data	x‘xxxx‘	Read 16 bits of RRAC_ID data from FlexIO/BIC interface.
40:87	Data	x‘0000_‘00000000’	All zeros to complete 88 bit SPI transaction.

The read data returned from the `rd_icb_data` register has 16 bits of FlexIO data returned on SPI data bits 24:39, as shown in *Figure 3-7* on page 106. The bit ordering is reversed on the Rambus interface.

Figure 3-7. FlexIO Read Data Mapping to SPI Read Data



SPI 0:7 is the Command, SPI 8:23 is the Address, SPI 24:87 is the Data

3.4 SPI Registers

As described in *Section 3.3 SPI Sequence Types* on page 99, registers that are accessed using simple reads or writes, or the Poll register (`icb_poll`), are considered to be directly accessible from the SPI interface without any additional sequencing steps. These are referred to as *SPI registers* in CBE documentation. SPI registers are defined only in this document, not in the *Cell Broadband Engine Registers* document, because SPI registers are not software accessible.

3.4.1 SPI Status Register

The SPI status register at SPI address x'0000' provides information to a System Controller regarding the internal status of the CBE processor. The SPI status register should be read when the attention signal is activated. Bits in the status register assist the System Controller in determining the cause of the attention signal activation. The status register can be read anytime by the System Controller to obtain the status of the CBE processor.

The SPI status register can also be written. Bit definitions for a write operation are different than those for a read operation. Write operations are used to clear the attention signal and to provide some control and diagnostic assistance for debug.

3.4.1.1 *Read SPI Status Register (`rd_spi_status`)*

The `rd_spi_status` register is seen by software when reading the SPI Status register.

SPI Address x'0000'

Type Read Only 32-bit Register

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bit	Definition	Description	Settings
0	Attention	Status of external Attention pin.	1 Active (CBE processor is issuing an Attention condition). 0 Inactive.
1	Machine Check	Status of any Machine Check condition generated within the CBE processor.	1 Active (machine check exists). 0 Inactive.
2	Quiesce	Status of any Quiesce condition in the CBE processor. Quiesce refers to the state in which the CBE processor is no longer executing instructions. It is the equivalent of an idle state.	1 Quiesce occurred. 0 No Quiesce.
3	Checkstop	Status of any Checkstop issued. Depending on the mask register setting in the FIR registers, external check stop can cause this bit to be '1'.	1 Check Stop condition. 0 No Check Stop condition.
4	Recoverable	Status of any recoverable errors.	1 Error occurred. 0 No Error occurred.
5	Reserved		
6	Thermal Event	Status of any Thermal Management Unit event, indicating that an action is needed to reduce the temperature of the chip.	1 Thermal event has occurred. 0 No thermal event.

Bit	Definition	Description	Settings
7	Livelock detection	Status of whether the one or more of the units on the EIB have detected a livelock condition. This bit gets cleared when livelocks are no longer present.	1 Livelock detected. 0 Livelock not detected.
8	FlexIO Reg Status	Status of the FlexIO register interface. This bit is set after the completion of a valid reg_reset operation. When active (1), the System Controller can access the FlexIO register interface.	1 Register interface active. 0 Register interface not available.
9	XIO Reg Status	Status of the XIO register interface. This bit is set after the completion of a valid reg_reset operation. When active (1), the System Controller can access the XIO register interface.	1 Register interface active. 0 Register interface not available.
10	POR Attention Req 0	Used by the POR logic to identify the cause for the activation of the attention signal during the POR sequence. If an Attention signal goes active and these bits are both zero, the POR logic did not cause the attention.	00 No attention. 01 FlexIO (RRAC) calibration request. 10 Configuration-ring data request. 11 Reserved.
11	POR Attention Req 1		
12:20	Reserved		
21	PPU Thread 0 Attention	Status of the PPU thread 0 Attention request signal.	1 Attention active. 0 Attention inactive.
22	PPU Thread 1 Attention	Status of the PPU thread 1 Attention request signal.	1 Attention active. 0 Attention inactive.
23:24	Reserved		
25	Reset Active	Status of the internal reset signal.	1 Internal reset active. 0 Internal reset is not active.
26	Reserved		
27	Core PLL Status	Status of the Core PLL lock signal.	1 Core PLL is locked. 0 Core PLL is unlocked.
28	FlexIO PLL Status	Status of the FlexIO PLL lock signal.	1 FlexIO PLL is locked. 0 FlexIO PLL is unlocked.
29	XIO PLL Status	Status of the XIO PLL lock signal.	1 XIO PLL is locked. 0 XIO PLL is unlocked.
30	0	This bit is constant 0.	Any value other than 0 is invalid.
31	1	This bit is constant 1.	Any value other than 1 is invalid.

3.4.1.2 *Write SPI Status Register (`wr_spi_status`)*

The `wr_spi_status` register is seen by software when writing the SPI Status register. Bits [10:13] are intended for lab debug and normally are not used in a production system.

SPI Address x'0000'

Type Write Only 32-bit Register

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bit	Definition	Description	Settings	
0	Clear ICB	Reset the Internal Configuration Bus (ICB) state machine. In normal conditions, there is no need to enable this bit as '1'.	1 0	Reset ICB. Do not reset ICB.
1:7	Reserved			
8	train_io	Indicate that the I/O calibration for the RAMBUS interface has completed.	1 0	I/O calibration complete. No action.
9	Reserved			
10	rrac_lock	Force the Pervasive Logic to override the pll_lock signal from the FlexIO. *(For lab debug; normally are not used in a production system)*	1 0	Activate FlexIO pll_lock signal. No action.
11	XIO_lock	Force the Pervasive Logic to override the pll_lock signal from the XIO. *(For lab debug; normally are not used in a production system)*	1 0	Activate XIO pll_lock signal. No action.
12	rrac reg_ready	Force the Pervasive Logic to override the FlexIO reg_ready signal from the BIC. *(For lab debug; normally are not used in a production system)*	1 0	Activate FlexIO reg_ready signal. No action.
13	XIO reg_ready	Force the Pervasive Logic to override the XIO reg_ready signal from the MIC. *(For lab debug; normally are not used in a production system)*	1 0	Activate XIO reg_ready signal. No action.
14:15	Reserved			
16	Quiesce MFC	Send quiesce request to all MFCs in the SPEs for livelock resolution mode.	1 0	Activate quiesce request to MFCs. Deactivate quiesce request to MFCs.
17	Quiesce PPSS	Send quiesce request to the PPSS in the PPE for livelock resolution mode.	1 0	Activate quiesce request to PPSS. Deactivate quiesce request to PPSS.
18	Thermal throttle PPE	Send thermal throttle request to PPE.	1 0	Activate throttle request to PPE. Deactivate throttle request to PPE.
19	Livelock resolution mode	Send livelock resolution mode (LRM) request to EIB.	1 0	Activate LRM to EIB. Deactivate LRM to EIB.
20:31	Reserved			

3.4.2 Write Configuration Ring (`wr_config_ring`)

The Configuration Ring is used to configure the CBE processor. This ring is defined as a write-only `wr_config_ring` register at SPI address x'0001'. The Configuration Ring can only be written by the System Controller during the POR sequence. The System Controller needs to read the SPI status register (*Section 3.4.1* on page 107) to determine when the Configuration Ring can be written. If bits [10:11] of the `spi_status` register is '10', the Configuration-Ring data can be written. See *Section 4 Configuration Ring* on page 119 for details on how to load the Configuration Ring.

3.4.3 ICB Poll Register (`icb_poll`)

The 8-bit `icb_poll` register is used to determine when an ICB read operation has completed. This register is described in *Section 3.3.3 Polling* on page 100.

SPI Address x'0002'

Type Read Only 8-bit Register

0	1	2	3	4	5	6	7

Bit	Definition	Description	Settings
0	Ready Indicator	Status of current ICB read operation.	1 Data is ready to be read out. 0 Data is still not ready yet.
1:7	Reserved		

3.4.4 Read CBE Chip ID (`rd_chip_id`)

The `rd_chip_id` register holds CBE Chip ID information.

SPI Address x'0003'

Type Read Only 32-bit Register

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bit	Definition	Description	Settings
0:3	Version	The CBE processor version number.	The current value is '0011'.
4:19	Part Number	The CBE processor part number.	Current part number is x'2740'.
20:30	Manufacturer ID	The CBE processor manufacturer ID.	For IBM-manufactured CBE processors, this ID is '00000100100'.
31	Constant	Always assigned as '1'.	

3.4.5 Read Serial Number Register(`rd_serial_num0`, `rd_serial_num1`)

The Read Serial Number registers—`rd_serial_num0` and `rd_serial_num1`—contain 48-bits of customer ID data provided by the customer. This register is divided into two registers; address x‘000A’ contains the most-significant 32-bits and address x‘000B’ the least-significant 16-bits. This pair of registers contains the same data as the serial_number MMIO register. The data in these registers is only valid after the POR sequence has completed because these registers are loaded from the Configuration Ring.

SPI Address x’000A’

Type Read Only 32-bit Register

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bit	Definition	Description	Settings
0:31	Customer ID	Customer ID [0:31] corresponds to fields s0:s31 in the serial_number MMIO register.	

SPI Address x’000B’

Type Read Only 32-bit Register

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bit	Definition	Description	Settings
0:15	Customer ID	Customer ID [32:47] corresponds to fields s32:s47 in the serial _number MMIO register.	
16:31	Reserved		

3.4.6 Read Voltage ID (`rd_VID`)

The voltage ID for the core is provided on each CBE processor to identify the optimum voltage setting for the part. The `rd_VID` register is read during the POR sequence. After the `rd_VID` register has been read, the system can adjust the core VDD for the CBE processor to this optimized voltage setting.

SPI Address x'000C'

Type Read Only 32-bit Register

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bit	Definition	Description	Settings
0:7	VID code	VID code for Voltage Regulation Module (VRM) setting.	Refer to the *Cell Broadband Engine Datasheet* for voltage associated with this 8 bit field.
8:31	Reserved		

3.4.7 Read Partial Good Register (`rd_partial_good`)

The `rd_partial_good` register indicates which SPEs are good. During CBE-processor manufacturing tests, manufacturing identifies bad SPEs on the chip. When a bad SPE is detected, manufacturing programs fuse bits on the CBE processor to identify the bad SPE(s). A '0' in the `rd_partial_good` register implies that the SPE is good. A '1' indicates that the SPE is bad.

Any SPEs marked as disabled in this register will be disabled internally. Specifying the corresponding SPE as enabled in Configuration Ring will not overwrite the effects of the fuse settings which appear in this register.

Because the `spe_available` MMIO register gets its value from the SPE Disable field on the Configuration Ring during POR, the external system controller must read the `rd_partial_good` register before the Configuration-Ring write has taken place in order to set the SPE Disable field to match the `rd_partial_good` register.

SPI Address x'000D'

Type Read Only 32-bit Register

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bit	Definition	Description	Settings
0	SPE 0 disabled	SPE 0 status.	1 SPE disabled. 0 SPE enabled.
1	SPE 1 disabled	SPE 1 status.	1 SPE disabled. 0 SPE enabled.
2	SPE 2 disabled	SPE 2 status.	1 SPE disabled. 0 SPE enabled.
3	SPE 3 disabled	SPE 3 status.	1 SPE disabled. 0 SPE enabled.
4	SPE 4 disabled	SPE 4 status.	1 SPE disabled. 0 SPE enabled.
5	SPE 5 disabled	SPE 5 status.	1 SPE disabled. 0 SPE enabled.
6	SPE 6 disabled	SPE 6 status.	1 SPE disabled. 0 SPE enabled.
7	SPE 7 disabled	SPE 7 status.	1 SPE disabled. 0 SPE enabled.
8:31	Reserved		

3.4.8 Read Linear Thermal Diode Calibration Register (`rd_lin_therm_diode`)

The `rd_lin_therm_diode` register contains the linear thermal diode calibration information that is recorded during manufacturing test calibration of the diode. 24 bits are provided for calibration information. See the *Cell Broadband Engine Datasheet* for more information on the linear thermal diode.

SPI Address x'000E'

Type Read Only 32-bit Register

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bit	Description	Setting
0:23	Linear Diode Calibration Data.	See the *Cell Broadband Engine Datasheet* for the Calibration Data format.
24:31	Reserved	

3.4.9 Read POR Status Register (`rd_por_status`)

The `rd_por_status` register contains the current state of the POR state machine. The System Controller reads this register to determine the state of the POR sequence.

SPI Address x'000F'

Type Read Only 32-bit Register

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

Bit	Definition	Description	Settings	
0	Error	Identifies if a error has occurred during the POR sequence.	1 0	Error or Invalid instruction encountered. No error.
1	Timeout	Indicates if the POR engine did not complete a POR instruction in the allotted time.	1 0	Timeout. No Timeout.
2:7	Reserved			
8	I/O Train	This bit indicates that I/O calibration is underway.	1 0	Calibration is active. Calibration is not active.
9	I/O Training Status	This bit is used in conjunction with bit [8] to assess the state of the I/Os. The POR state machine sets this bit once the System Controller has indicated that I/O calibration is done by setting wr_spi_status[8].	1 0	I/O calibration complete. I/O calibration not complete.
10	Configuration Active	Indicates that the POR state machine on the CBE processor is requesting Configuration-Ring data to be scanned in.	1 0	Configuration active. No configuration.
11	Configuration Complete	Indicates that the POR state machine on the CBE processor has recognized that the Configuration Ring has been scanned in (has seen the leading 1 or "start" bit).	1 0	Configuration complete. Configuration not complete.
12	FlexIO PLL Locked	Indicates the state of the FlexIO PLL lock signal received by the POR state machine.	1 0	FlexIO PLL Locked. FlexIO PLL not Locked.
13	XIO PLL Locked	Indicates the state of the XIO PLL lock signal received by the POR state machine.	1 0	XIO PLL Locked. XIO PLL not Locked.
14	FlexIO Reg Status	Status of the FlexIO register interface. When active, the System Controller can access the FlexIO register interface.	1 0	FlexIO register interface active. FlexIO register interface not available.
15	XIO Reg Status	Status of the XIO register interface. When active, the System Controller can access the XIO register interface.	1 0	XIO register interface active. XIO register interface not available.
16	Reserved			
17	Credit Enable	This signal indicates the state of the credit enable signal that is sent to the EIB.	1 0	Credit enable active. Credit enable inactive.
18	Reserved			
19	Reserved			
20	Wait State	The POR state machine on the CBE processor is in a wait state. If set, the CBE processor requires external system controller intervention to continue.	1 0	CBE processor in a wait state. CBE processor not wait state.

Bit	Definition	Description	Settings	
21	Reserved			
22	POR Complete	This bit is active when the POR state machine has completed its entire sequence. POR is complete.	1 0	POR complete. POR not complete.
23	Phase 2	This bit indicates that the POR state machine has entered phase 2 initialization.	1 0	POR is in phase 2. POR is not in phase 2.
24:31	Reserved			

3.4.10 Read ICB Data Register (`rd_icb_data`)

The `rd_icb_data` register at SPI address x'0010' is used to provide up to 64 bits of read data for an ICB read transaction. The transfer size for this register is always 64 bits, but the amount of data depends on the definition of the register. See *Section 3.3.4.3 ICB Read Example* on page 102 for usage information on this register.

Reading the `rd_icb_data` register accesses the ICB bus to either the MIC or BIC logic. If the physical location of the data is exactly known, the exact number of bits transferred on ICB will be known, and the transfer operation can be stopped after the length of the expected data. For example if software accesses a 12-bit register from MIC logic and all bits are known to be left-aligned, the read can be stopped after 12 bits.

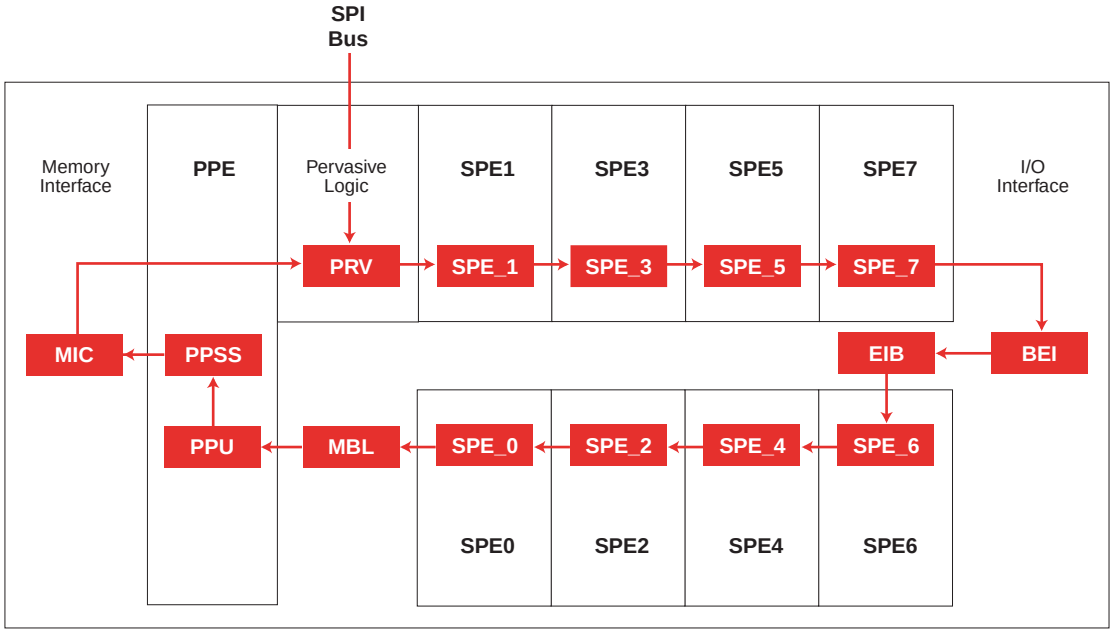
4. Configuration Ring

The Configuration Ring is a series of bits that must be loaded into the CBE processor during the Power-On Reset (POR) sequence. The bits are loaded through the Serial Peripheral Interface (SPI) into internal latches which remain static until the CBE processor is rebooted. The bits consist of configuration data for CBE processor functional units that either must remain static or that can be changed due to system requirements. This section describes the bit-field definitions and default values for the Configuration Ring.

4.1 Load Path

Figure 4-1 shows the CBE-processor path through which the Configuration-Ring bits are loaded. The ring is not a scan chain and does not shift any bits out when loaded from the SPI interface.

Figure 4-1. Configuration-Ring Path



BEI	Cell Broadband Engine Interface
EIB	Element Interconnect Bus
MBL	MIC Bus Logic
MIC	Memory Interface Controller
PPE	PowerPC Processor Element
PPSS	PowerPC Processor Storage Subsystem
PPU	PowerPC Processor Unit
PRV	Pervasive Logic
SPE	Synergistic Processor Element
SPI	Serial Peripheral Interface

4.2 Bit Descriptions

Table 4-1 describes the Configuration-Ring bit fields for both single-CBE-processor and dual-CBE-processor systems. The red[1] rows indicate the beginning of bit fields for a particular function unit (the functional units relevant to the Configuration Ring are shown in *Figure 4-1* on page 119). Bit fields for some functional units are connected in reverse order, such that the least-significant latch bit is connected to the most-significant Configuration-Ring bit for that bit-field. This difference is reflected in *Table 4-1* by reversing the values in the Bit Offset column.

The default values for reserved fields shown in this section are recommended values that are loaded during the Configuration-Ring write portion of the POR sequence. These values are for first-generation CBE processors. Future revisions of the CBE processor may require different values in some of these fields. Contact your SCEI, Toshiba, or IBM representative for updates to these fields. The default values listed for the defined fields are shown here as examples only. Users of this document may change the values in these fields as required for their system configurations.

There are 2697 bits in the ring—bits [0:2696]. When loading the ring through the SPI interface, shift bit [2696] in first. After all bits are loaded, the CBE processor is clocked by hardware to set the latches. For a description of how to write the Configuration Ring using the SPI interface, see *Section 3 Serial Peripheral Interface* on page 91.

For details about base-address configuration, see the Resource Allocation Management chapter of the *Cell Broadband Engine Programming Handbook*.

Table 4-1. Configuration Ring Fields (Page 1 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
Pervasive Logic (PRV) Bits					
0:1	2	x‘0’	x‘0’	x‘0’	Reserved.
SPE_1 Bits					
2: 10	11	x‘000’	x‘000’	x‘000’	Reserved.
11: 25	15	x‘0000’	x‘0000’	x‘2000’	SPE1 MC_BASE. A 15-bit register for the MC_BASE address.
26: 40	15	x‘4000’	x‘7FFE’	x‘7FFE’	SPE1 MC_COMP_EN. A 15-bit register to specify the memory controller size.
41: 50	10	x‘380’	x‘380’	x‘380’	SPE1 IOIF1_COMP_EN. A 10-bit register to specify the IOIF1 size.
51: 186	136	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000812’	Reserved.
187: 205	19	x‘40000’	x‘40000’	x‘60000’	SPE1 BE_MMIO_Base. A 19-bit register for the SPE1 base address.
206: 209	4	x‘0’	x‘0’	x‘1’	SPE1 unit CBE Node ID.

1. Red when viewed on-screen, or gray when printed on a black-and-white printer.

Table 4-1. Configuration Ring Fields (Page 2 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
210: 212	3	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	SPE1 Synergistic processor element (SPE) ID.
213: 223	11	x‘1B0’	x‘1B0’	x‘1B0’	Reserved.
SPE_3 Bits					
224: 232	9	x‘000’	x‘000’	x‘000’	Reserved.
233: 247	15	x‘0000’	x‘0000’	x‘2000’	SPE3 MC_BASE. A 15-bit register for the MC_BASE address.
248: 262	15	x‘4000’	x‘7FFE’	x‘7FFE’	SPE3 MC_COMP_EN. A 15-bit register to specify the memory controller size.
263: 272	10	x‘380’	x‘380’	x‘380’	SPE3 IOIF1_COMP_EN. A 10-bit register to specify the IOIF1 size.
273: 408	136	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000812’	Reserved.
409: 427	19	x‘40000’	x‘40000’	x‘60000’	SPE3 BE_MMIO_Base. A 19-bit register for the SPE3 base address.
428: 431	4	x‘0’	x‘0’	x‘1’	SPE3 unit CBE Node ID.
432: 434	3	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	SPE3 Synergistic processor element (SPE) ID.
435: 445	11	x‘1B0’	x‘1B0’	x‘1B0’	Reserved.
SPE_5 Bits					
446: 454	9	x‘000’	x‘000’	x‘000’	Reserved.
455: 469	15	x‘0000’	x‘0000’	x‘2000’	SPE5 MC_BASE. A 15-bit register for the MC_BASE address.
470: 484	15	x‘4000’	x‘7FFE’	x‘7FFE’	SPE5 MC_COMP_EN. A 15-bit register to specify the memory controller size.

Table 4-1. Configuration Ring Fields (Page 3 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
485: 494	10	x‘380’	x‘380’	x‘380’	SPE5 IOIF1_COMP_EN. A 10-bit register to specify the IOIF1 size.
495: 630	136	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000812’	Reserved.
631: 649	19	x‘40000’	x‘40000’	x‘60000’	SPE5 BE_MMIO_Base. A 19-bit register for the SPE5 base address.
650: 653	4	x‘0’	x‘0’	x‘1’	SPE5 unit CBE Node ID.
654: 656	3	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	SPE5 Synergistic processor element (SPE) ID.
657: 667	11	x‘1B0’	x‘1B0’	x‘1B0’	Reserved.
SPE_7 Bits					
668: 676	9	x‘000’	x‘000’	x‘000’	Reserved.
677: 691	15	x‘0000’	x‘0000’	x‘2000’	SPE7 MC_BASE. A 15-bit register for the MC_BASE address.
692: 706	15	x‘4000’	x‘7FFE’	x‘7FFE’	SPE7 MC_COMP_EN. A 15-bit register to specify the memory controller size.
707: 716	10	x‘380’	x‘380’	x‘380’	SPE7 IOIF1_COMP_EN. A 10-bit register to specify the IOIF1 size.
717: 852	136	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000812’	Reserved.
853: 871	19	x‘40000’	x‘40000’	x‘60000’	SPE7 BE_MMIO_Base. A 19-bit register for the SPE7 base address.
872: 875	4	x‘0’	x‘0’	x‘1’	SPE7 unit CBE Node ID.
876: 878	3	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	SPE7 Synergistic processor element (SPE) ID.
879: 889	11	x‘1B0’	x‘1B0’	x‘1B0’	Reserved.

Table 4-1. Configuration Ring Fields (Page 4 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
Cell Broadband Engine Interface (BEI) Unit Bits					
891: 890[1]	2	‘00’	‘00’	‘00’	Reserved.
895: 892[1]	4	x‘0’	x‘0’	x‘1’	BIF-Unit CBE Node ID. BIF unit CBE Node ID for this CBE processor. Multiple CBE processors connected by a BIF must have different values. This value must be consistent with the node ID values configured in the other units in the same CBE processor.
917: 896[1]	22	x‘200005’	x‘200005’	x‘300005’	BEI BE_MMIO_Base. This value defines the most-significant 22 bits of the real address used to access BEI and EIB MMIO registers. The most-significant 19 bits of this value should be consistent with the value loaded into the BE_MMIO_Base registers for the SPEs, PPE, MIC, and PRV Configuration-Ring fields. The least-significant 3 bits of this value should be ‘101’ so that the BEI register offset in the CBE MMIO space matches *Table A-3 CBE-Processor Memory Map* on page 148.
920: 918[1]	3	’110’	’110’	’110’	Reserved.
923: 921[1]	3	‘011’	‘011’	‘011’	Reserved.
935: 924[1]	12	x‘F80’	x‘F80’	x‘F80’	IOIF1 Base Address Mask. With the IOIF1 base address, these bits define the initial range of addresses mapped to IOIF1. These bits get copied to IOC_BaseAddrMask1 MMIO register bits [0,22:32] during POR when clocks are started.
957: 936[1]	22	x‘240000’	x‘240000’	x‘340000’	IOIF1 Base Address and Replacement. With the IOIF1 address mask, these bits define the initial range of addresses mapped to IOIF1 and the upper IOIF1 address bits used for the outbound read or write. These bits get copied to IOC_BaseAddr0 MMIO register bits [22:32, 53:63] during POR when clocks are started.
969: 958[1]	12	x‘F80’	x‘000’	x‘000’	IOIF0 Base Address Mask. With the IOIF0 base address, these bits define the initial range of addresses mapped to IOIF0. These bits get copied to IOC_BaseAddrMask0 MMIO register bits [0, 22:32] during POR when clocks are started.
991: 970[1]	22	x‘280000’	x‘000000’	x‘000000’	IOIF0 Base Address and Replacement.
993: 992[1]	2	x‘0’	x‘0’	x‘0’	Reserved.
995: 994[1]	2	x‘10’	x‘11’	AC0 Configuration	AC0 Configuration. Address Concentrator 0 (AC0) configuration mode. Enables and disables parts of AC0, depending on how many CBE processors are in the system and where the single system wide AC0 resides: ‘00’ or ‘01’ AC0 off-chip (2 CBE processor configuration, non-AC0 CBE processor). ‘10’ AC0 on-chip, no off-chip AC1 (one-CBE-processor configuration). ‘11’ AC0 on-chip, off-chip AC1 present (two-CBE-processor configuration, AC0 CBE processor).

Table 4-1. Configuration Ring Fields (Page 5 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
1000: 996[1]	5	'00100'	'00010'	'00010'	BIF/IOIF0 RX Configuration. Number of Rambus ASIC cell (FlexIO) receive blocks in BIF/IOIF0: '00000' 0 blocks. '10000' 1 block. '01000' 2 blocks. '00100' 3 blocks. '00010' 4 blocks. '00001' 5 blocks.
1006: 1001[1]	6	'000100'	'000100'	'000100'	BIF/IOIF0 TX Configuration. Number of Rambus ASIC cell (FlexIO) send blocks in BIF/IOIF0: '000000' 0 blocks. '100000' 1 block. '010000' 2 blocks. '001000' 3 blocks. '000100' 4 blocks. '000010' 5 blocks. '000001' 6 blocks.
1012: 1007[1]	6	'000000'	'000000'	'000000'	Reserved.
1013	1	1	0	0	BIF/IOIF0 Coherency Mode. BIF/IOIF0 operation mode: 0 Broadband Processor Interface (BIF). 1 IOIF.
1016: 1014[1]	3	'000'	'000'	'000'	Reserved.
1019: 1017[1]	3	'100'	'001'	'001'	BIF/IOIF0 Reorder. BIF/IOIF0 I/O reorder mode for transmit When two CBE processors are connected to each over slice 0, the sending side needs to reverse the order of the groups of 24 bits that feed each TX FlexIO block. Assuming the bits are assigned to blocks A, B, C, D, E, and F in groups of 24 bits starting from bit [0] through bit [143] (that is, A=[0:23], B=[24:47],...), the following values are valid for this field. All other values produce unpredictable results. In all cases, blocks E and F remain connected to themselves and are not reordered. Also, no bit reversal within the 24 bits is performed as the FlexIO TX and RX bits are already placed in a reverse order during manufacturing. Reserved adjacent (following) bit for expansion of this configuration item: '100' No reordering (A->A, B->B, C->C, D->D). '010' 3 block reordering (C->A, B->B, A->C, D->D). '001' 4 block reordering (D->A, C->B, B->C, A->D). All encodings not listed are invalid.
1035: 1020[1]	16	x'0000'	x'0000'	x'0000'	Reserved.

Table 4-1. Configuration Ring Fields (Page 6 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
1038: 1036[1]	3	’100’	’100’	’100’	IOIF1 Reorder. IOIF1 I/O reorder mode for transmit. When two CBE processors are connected to each over IOIF1, the sending side needs to reverse the order of the groups of 24 bits that feed each TX FlexIO block. Assuming the bits are assigned to blocks A and B in groups of 24 bits starting from bit [0] through bit [47] (that is A=[0:23], B=[24:47]), the following values are valid for this field. All other values produce unpredictable results. Also, no bit reversal within the 24 bits is performed as the FlexIO TX and RX bits are already placed in a reverse order when bonded out on the die and module. ‘100’ No reordering (A->A, B->B). ‘010’ No reordering (A->A, B->B). ‘001’ 2 block reordering (B->A, A->B). All encodings not listed are invalid.
1039	1	1	1	1	Reserved.
1041: 1040[1]	2	‘10’	‘10’	‘10’	IOIF1 RX Configuration. Number of Rambus ASIC cell (FlexIO) receive (RX) blocks in IOIF1: ‘00’ 0 blocks. ‘10’ 1 block. ‘01’ 2 blocks. All encodings not listed are invalid.
1043: 1042[1]	2	‘10’	‘10’	‘10’	IOIF1 TX Configuration. Number of Rambus ASIC cell (FlexIO) transmit (TX) blocks in IOIF1: ‘00’ 0 blocks. ‘10’ 1 block. ‘01’ 2 blocks. All encodings not listed are invalid.
1075: 1044[1]	32	x‘00000200’	x‘00000200’	x‘00000200’	FlexIO PLL Configuration. This field connects directly to the rc_pll_config input of the RRAC. For this field, bit [1044] corresponds to rc_pll_config[31], and bit [1075] corresponds to rc_pll_config[0]. See the Rambus specification for details.
1077: 1076[1]	2	‘00’	‘00’	‘00’	Reserved.
Element Interconnect Bus (EIB) Unit Bits					
1078: 1079	2	‘00’	‘00’	‘00’	Reserved.
1080	1	0	0	0	AC0 Livelock Response Control: 0 AC0 runs in single-step mode (one command per PAAM window) when the CBE processor is in livelock response mode. 1 AC0 ignores livelock response mode.
1084: 1081[1]	4	x‘0’	x‘0’	x‘1’	EIB unit CBE Node ID. This field is scanned in reverse order.
1085	1	0	1	0	AC1 Configuration: 0 No off-chip AC1 present. 1 Off-chip AC1 present.

Table 4-1. Configuration Ring Fields (Page 7 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
1086	1	1	1	0	AC0 configuration: 0 AC0 off-chip (other CBE processor is master). 1 AC0 on-chip (this CBE processor is master).
1087: 1090	4	‘0010’	‘0010’	‘1000’	AC0 command credits: ‘0010’ AC0 is on-chip. ‘1000’ AC0 is off- chip.
1091: 1112	22	x‘200000’	x‘200000’	x‘300000’	LBAR0_cfg. Used to set the Local Base Address Register 0 (EIB_LBAR0) from the Configuration Ring. When clocks start, the content of LBAR0_cfg is copied to EIB_LBAR0.
1113: 1134	22	x‘3FFFF8’	x‘3FFFF8’	x‘3FFFF8’	LBAMR0_cfg. Used to set the Local Base Address Mask Register 0 (EIB_LBAMR0) from the Configuration Ring. When clocks start, the content of LBAMR0_cfg is copied to EIB_LBAMR0.
1135: 1137	3	’011’	’011’	’011’	Reserved.
1138	1	‘0’	‘0’	‘0’	AC1 Livelock Response Control: 0 AC1 disables all Local Address Ranges when the CBE processor is in livelock response mode. 1 AC1 ignores livelock response mode.
1139: 1140	2	‘0’	‘0’	‘0’	Reserved.
SPE_6 Bits					
1141: 1149	9	x‘000’	x‘000’	x‘000’	Reserved.
1150: 1164	15	x‘0000’	x‘0000’	x‘2000’	SPE6 MC_BASE. A 15-bit register for the MC_BASE address.
1165: 1179	15	x‘4000’	x‘7FFE’	x‘7FFE’	SPE6 MC_COMP_EN. A 15-bit register to specify the memory controller size.
1180: 1189	10	x‘380’	x‘380’	x‘380’	SPE6 IOIF1_COMP_EN. A 10-bit register to specify the IOIF1 size.
1190: 1325	136	x‘80000000000000000000000000000802’	x‘80000000000000000000000000000802’	x‘80000000000000000000000000000812’	Reserved.
1326: 1344	19	x‘40000’	x‘40000’	x‘60000’	SPE6 BE_MMIO_Base. A 19-bit register for the SPE6 base address.
1345: 1348	4	x‘0’	x‘0’	x‘1’	SPE6 unit CBE Node ID.
1349: 1351	3	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	SPE6 Synergistic processor element (SPE) ID.

Table 4-1. Configuration Ring Fields (Page 8 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
1352: 1362	11	x‘1B0’	x‘1B0’	x‘1B0’	Reserved.
SPE_4 Bits					
1363: 1371	9	x‘000’	x‘000’	x‘000’	Reserved.
1372: 1386	15	x‘0000’	x‘0000’	x‘2000’	SPE4 MC_BASE. A 15-bit register for the MC_BASE address.
1387: 1401	15	x‘4000’	x‘7FFE’	x‘7FFE’	SPE4 MC_COMP_EN. A 15-bit register to specify the memory controller size.
1402: 1411	10	x‘380’	x‘380’	x‘380’	SPE4 IOIF1_COMP_EN. A 10-bit register to specify the IOIF1 size.
1412: 1547	136	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000812’	Reserved.
1548: 1566	19	x‘40000’	x‘40000’	x‘60000’	SPE4 BE_MMIO_Base. A 19-bit register for the SPE4 base address.
1567: 1570	4	x‘0’	x‘0’	x‘1’	SPE4 unit CBE Node ID.
1571: 1573	3	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	SPE4 Synergistic processor element (SPE) ID.
1574: 1584	11	x‘1B0’	x‘1B0’	x‘1B0’	Reserved.
SPE_2 Bits					
1585: 1593	9	x‘000’	x‘000’	x‘000’	Reserved.
1594: 1608	15	x‘0000’	x‘0000’	x‘2000’	SPE2 MC_BASE. A 15-bit register for the MC_BASE address.
1609: 1623	15	x‘4000’	x‘7FFE’	x‘7FFE’	SPE2 MC_COMP_EN. A 15-bit register to specify the memory controller size.
1624: 1633	10	x‘380’	x‘380’	x‘380’	SPE2 IOIF1_COMP_EN. A 10-bit register to specify the IOIF1 size.
1634: 1769	136	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000812’	Reserved.
1770: 1788	19	x‘40000’	x‘40000’	x‘60000’	SPE2 BE_MMIO_Base. A 19-bit register for the SPE2 base address.

Table 4-1. Configuration Ring Fields (Page 9 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
1789: 1792	4	x‘0’	x‘0’	x‘1’	SPE2 unit CBE Node ID.
1793: 1795	3	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	SPE2 Synergistic processor element (SPE) ID.
1796: 1806	11	x‘1B0’	x‘1B0’	x‘1B0’	Reserved.
SPE_0 Bits					
1807: 1815	9	x‘000’	x‘000’	x‘000’	Reserved.
1816: 1830	15	x‘0000’	x‘0000’	x‘2000’	SPE0 MC_BASE. A 15-bit register for the MC_BASE address.
1831: 1845	15	x‘4000’	x‘7FFE’	x‘7FFE’	SPE0 MC_COMP_EN. A 15-bit register to specify the memory controller size.
1846: 1855	10	x‘380’	x‘380’	x‘380’	SPE0 IOIF1_COMP_EN. A 10-bit register to specify the IOIF1 size.
1856: 1991	136	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000802’	x‘80000000000 0000000000000 0000000812’	Reserved.
1992: 2010	19	x‘40000’	x‘40000’	x‘60000’	SPE0 BE_MMIO_Base. A 19-bit register for the SPE0 base address.
2011: 2014	4	x‘0’	x‘0’	x‘1’	SPE0 unit CBE Node ID.
2015: 2017	3	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	‘000’ SPE0 ‘001’ SPE1 ‘010’ SPE2 ‘011’ SPE3 ‘100’ SPE4 ‘101’ SPE5 ‘110’ SPE6 ‘111’ SPE7	SPE0 Synergistic processor element (SPE) ID.
2018: 2028	11	x‘1B0’	x‘1B0’	x‘1B0’	Reserved.
MIC Bus Logic (MBL) Bits					
2029: 2032	4	x‘8’	x‘8’	x‘8’	Reserved.

Table 4-1. Configuration Ring Fields (Page 10 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
2033: 2048	16	x‘0000’	x‘0000’	x‘FFF8’	Address Start. The MIC Address Space Start and End registers are sized at 16 bits to cover a minimum possible size for the MIC memory range granularity of 64 MB. These 16 bits correspond to the upper 16 bits of the 42-bit EIB Address bits [22:37]. These two registers are used to determine whether an address associated with an incoming EIB command is within the address space of the MIC. The MIC Address Space Start register contains the 2’s-complemented starting address of the MIC address space range. The MIC Address Space End register contains the 2’s-complemented address of the next block following the ending address of the MIC address space. Example: if the address starts at 0: 0 MB / 64 MB = 0 = ‘0000 0000 0000 0000’ ‘0000 0000 0000 0000’. ‘1111 1111 1111 1111’ (one’s complement (invert)). ‘0000 0000 0000 0000’ (add 1 (two’s complement)). x‘0000’ is the value to program.
2049: 2064	16	x‘FFF8’	x‘FFF8’	x‘FFF0’	Address End. The ending address of the MIC as described in the previous description (MIC Address Space Start). Example: if the MIC address space ends at 512 MB: 512 MB / 64 MB = 8 = ‘0000 0000 0000 1000’ ‘0000 0000 0000 1000’. ‘1111 1111 1111 0111’ (one’s complement (invert)). ‘1111 1111 1111 1000’ (add 1 (two’s complement)). x‘FFF8’ is the value to program.
2065: 2094	30	x‘20000509’	x‘20000509’	x‘30000509’	PRV BE_MMIO_Base. This register contains 30 bits that are compared to the incoming EIB Address bits [22:51] to determine whether the incoming EIB command is an MMIO Command to the Pervasive Logic. The most-significant 19 bits of this value should be consistent with the value loaded into the BE_MMIO_Base registers for the SPEs, PPE, MIC, and BEI Configuration-Ring fields.
2095: 2124	30	x‘2000050A’	x‘2000050A’	x‘3000050A’	MIC BE_MMIO_Base. This register contains 30 bits that are compared to the incoming EIB Address bits [22:51] to determine whether the incoming EIB command is an MMIO command to the MIC logic. The most-significant 19 bits of this value should be consistent with the value loaded into the BE_MMIO_Base registers for the SPEs, PPE, BEI, and PRV Configuration-Ring fields.
2125: 2128	4	x‘3’	x‘3’	x‘3’	Reserved.
2129: 2132	4	x‘0’	x‘0’	x‘1’	MIC unit CBE Node ID.
2133: 2134	2	x‘0’	x‘0’	x‘0’	Reserved.
PowerPC Processor Unit (PPU)) Bits					
2135: 2143	9	x‘000’	x‘000’	x‘000’	Reserved.

Table 4-1. Configuration Ring Fields (Page 11 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
2144: 2151	8	x‘00’	x‘00’	x‘01’	PIR_defn. Defines the setting of bits [23:30] of the Processor Identification Register (PIR). Each PPE contains one PIR register. The PIR is used for processor differentiation in multiprocessor systems. In a single PPE (single CBE processor) system, this register would be set to zero. With multiple PPEs (multiple CBE processors) in the system, the register would be set uniquely in each PPE. That is, in a system with two PPEs, one physical register would be set to zero and the register in the other processor would be set to a one. The following encodings are valid: x‘00’ CBE processor 0. x‘01’ CBE processor 1. All encodings not listed are invalid.
2152: 2164	13	x‘0000’	x‘0000’	x‘0000’	Reserved.
2165: 2204	40	x‘0000000000’	x‘0000000000’	x‘0000000000’	PPE SReset Vector. System reset interrupt address for PPE thread 0. If SPR register HID1[19] = ’0’, then set the system reset interrupt address for PPE thread 0 to the value specified here. This value needs to be defined by the system, but is shown here as all zeros. If HID1[19] = ’1’, then the system reset interrupt address for thread 0 will be x‘100’ and the value loaded here from the Configuration Ring will be ignored. The value seen in this 40-bit hex string is the desired physical address (by the PPE) shifted left by two bits. For example, if HID1[19] = ‘0’, and the desired system reset address to jump to is 0x’24000000100’, the value of this hex string would be 0x’900000040’.
2205: 2428	224	x‘00000000000000000000000080000000000000000000000000000000’	x‘00000000000000000000000080000000000000000000000000000000’	x‘00000000000000000000000080000000000000000000000000000000’	Reserved.
PowerPC Processor Storage Subsystem (PPSS)) Bits					
2429: 2489	61	x‘0010000000000800’	x‘0010000000000800’	x‘0010000000000801’	Reserved.
2490	1	1	1	1	L2 Livelock Indication enable: 0 Disable Livelock Indication logic. 1 Enable Livelock Indication logic to help recover from possible system livelock/starvation (default).
2491: 2534	44	x‘000000F8000’	x‘000000F8000’	x‘000000F8000’	Reserved.
2535: 2538	4	x‘0000’	x‘0000’	x‘0001’	PPE unit CBE Node ID. The following encodings are valid: x‘0’ CBE processor 0. x‘1’ CBE processor 1. All encodings not listed are invalid.
2539: 2547	9	x‘0D8’	x‘0D8’	x‘0D8’	Reserved.

Table 4-1. Configuration Ring Fields (Page 12 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
2548: 2577	30	x‘20000500’	x‘20000500’	x‘30000500’	PPE BE_MMIO_Base. A 19-bit register for the PPE base address. Also known as the PPE MMIO Address Space Range register.
2578: 2584	7	x‘47’	x‘47’	x‘47’	Reserved.
2585	1	x‘1’	x‘1’	x‘1’	2 Token Decode for NCU Store: PPSS requests 2 tokens for NCU store to memory when Resource Allocation is enabled. 1 PPSS requests two tokens. 0 PPSS requests one token. If Resource Allocation is disabled, this bit is “don’t care”. NCU store to IOIF space requires only one token, regardless of the setting of this bit.
2586: 2601	16	x‘0000’	x‘0000’	x‘4000’	Reserved.
2602	1	1	1	1	NCU Livelock Indication enable: 0 Disable Livelock Indication sourced by NCU. 1 Enable Livelock Indication sourced by NCU (default).
2603	1	1	1	1	PPE Livelock Indication enable: 0 Disable Livelock Indication sourced by PPE. 1 Enable Livelock Indication sourced by PPE (default).
2604: 2605	2	‘00’	‘00’	‘00’	Reserved.
Memory Interface Controller (MIC) Bits					
2621: 2606[1]	16	x‘0000’	x‘0000’	x‘0000’	XIO PLL (`YO_RQ_CTM`) Configuration, Lower Half. Rambus phase locked loop (PLL) configuration data This register is scanned in reverse order. The default value is supplied by Rambus and is system dependent. Scanning this register in reverse order causes bit [0] to be the first bit out, which Rambus interprets as the LSB (Rambus uses Little Endian numbering). This register is the least-significant half word.
2637: 2622[1]	16	x‘9CFC’	x‘9CFC’	x‘9CFC’	XIO PLL (`YO_RQ_CTM`) Configuration, Upper Half. This register is the most-significant half word and is also scanned in reverse order so that bit [16] is the first bit out and bit [31] is the last bit out. Rambus interprets bit[31] as the MSb.
2638: 2641	4	x‘0’	x‘0’	x‘0’	Reserved.
Pervasive Logic (PRV) Bits					
2642: 2647	6	x‘1F’	x‘1F’	x‘1F’	cfg_TO. Thermal Overload Temperature. The value in this field is the encoded temperature level which will cause the Thermal Overload signal to be asserted and the clocks stopped. (Suggested value is x‘1F’). Value of x‘00 disables the thermal overload protection feature.
2648: 2674	27	x‘0288018’	x‘0288018’	x‘0288018’	Reserved.

Table 4-1. Configuration Ring Fields (Page 13 of 13)

Bit Offset	Number of Bits	Single-CBE-Processor Default Value	Dual-CBE-Processor BIF-Mode Default Value		Description
			CBE0	CBE1	
2675	1	1	1	1	Pervasive-Logic Livelock Indication Enable: 0 Livelock Indication Disable. 1 Livelock Indication Enable (Default).
2676: 2685	10	x‘000’	x‘000’	x‘000’	Reserved.
2693: 2686[1]	8	‘00000000’	‘00000000’	‘00000000’	SPE Disable. These signals are used to enable or disable SPEs. The settings in this field get copied into bits 24:31 of the SPE_Available register. SPE disable is a one hot, 8-bit latch. A ‘1’ in positions 0 through 7 of the latch results in the corresponding SPE being disabled. Configuration ring bit [2686] corresponds to bit [7] (SPE 7), and ring bit [2693] corresponds to bit [0] (SPE0).
2696: 2694[1]	3	‘000’	‘000’	‘000’	Reserved.

1. Ring Bit Offset range for this function is reversed to indicate that the bits for it are reversed when writing this value. The logical value is shown in the Default Value columns.

5. Signal Descriptions

This chapter describes the CBE processor's external signals and power-related pins. Signal names are in uppercase letters. An active-low signal will be asserted when tied to ground. Active-low signals have an overbar on the signal name, as in $\overline{\text{SIGNAL\_NAME}}$. Differential pairs append "N" on the name of the negative signal and do not append anything on the name of the positive signal. All voltages are nominal; see the *Cell Broadband Engine Datasheet* for voltage specifications.

5.1 Signal Groups

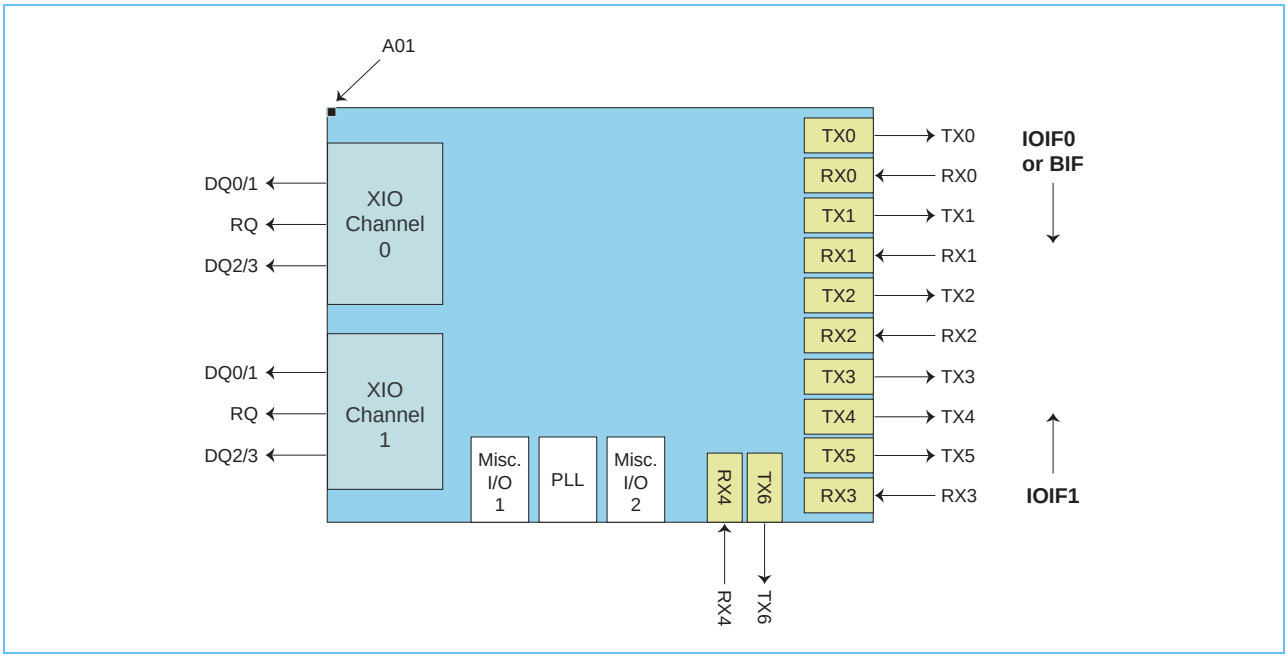
The signals are grouped as follows:

- **FlexIO Interface**—These signals provide a flexible chip-to-chip interconnect that can be configured as one or two IOIF-protocol interfaces or as one BIF-protocol interface and one IOIF-protocol interface, which combined provide up to 35 GBps of transmit bandwidth and 25 GBps of receive bandwidth. The data rate is 5 Gbps per differential pair, based on a 500 MHz reference clock. The physical layer interconnect for this interface is the Rambus FlexIO external I/O channels, formerly called the Redwood Rambus Access Cell (RRAC) channels.
- **FlexIO Power Supplies and References**—These pins provide power supply and reference voltages for the FlexIO interface.
- **XDR Memory Interface: Channel 0**—These signals provide a connection to Rambus XDR DRAM devices. The capacity of the channel is configureable using various bit-widths of XDR DRAMs. The bandwidth of the XDR channel is 12.8 GBps. The data rate is 3.2 Gbps per differential pair, based on a 400 MHz reference clock.
- **XDR Memory Serial Interface: Channel 0**—These signals provide a low-speed serial interface to the XDR DRAM devices which is used for initialization.
- **Memory Interface (XIO) Power Supplies and References: Channel 0—**These pins provide power supply and reference voltages for the XIO cell.
- **XDR Memory Interface: Channel 1—**These signals provide a connection to Rambus XDR DRAM devices. The capacity of the channel is configureable using various bit-widths of XDR DRAMs. The bandwidth of the XDR channel is 12.8 GBps. The data rate is 3.2 Gbps per differential pair, based on a 400 MHz reference clock.
- **XDR Memory Serial Interface: Channel 1**—These signals provide a low-speed serial interface to the XDR DRAM devices which is used for initialization.
- **Memory Interface (XIO) Power Supplies and References: Channel 1—**These pins provide power supply and reference voltages for the XIO cell.
- **Serial Peripheral Interface (SPI)**—These signals provide a serial interface used for CBE processor initialization and status monitoring.
- **Miscellaneous IO—**These are miscellaneous status and control signals.
- **Miscellaneous Test IO**—These are miscellaneous signals used only for test and debug.
- **Power Supply**—These pins provide the main power for the CBE processor.

5.2 Input/Output-Signal Layout

Figure 5-1 shows the general layout of the CBE processor's I/O blocks and signal groups. This view is from the top of the CBE processor (that is, live bug view) and corresponds to the layout you would see if looking at a card with a CBE processor attached.

Figure 5-1. CBE Module Footprint, Top View (Live Bug)



5.3 Signal Descriptions

5.3.1 FlexIO Interface

The FlexIO interface provides seven transmit bytes and five receive bytes of Rambus FlexIO (formerly known as RRAC) channel interface. Each differential pair carries 5.0 Gbps (2.5 Gbps in half-rate mode) of data at Differential Rambus Signaling Level (DRSL); see the Rambus documentation cited in the *Preface* on page 11 for details about DRSL. Each Rambus channel is eight bits wide and has its own differential data clock. The clock frequency is 500 MHz. At the physical layer, the FlexIO interface performs calibration during the Power-On Reset (POR) sequence to adjust the driver impedance and output levels and to align the data bits for the 8-bit channel with the data clock.

Table 5-1 on page 135 lists the FlexIO interface signals.

Table 5-1. FlexIO Interface Signals (Page 1 of 2)

Signal Name	Description
RX4_RX[7:0] RX4_RXN[7:0]	FlexIO Receive Channel, byte 4
RX4_RXCLK RX4_RXCLKN	FlexIO Receive Channel Clock for byte 4
RX3_RX[7:0] RX3_RXN[7:0]	FlexIO Receive Channel, byte 3
RX3_RXCLK RX3_RXCLKN	FlexIO Receive Channel Clock for byte 3
RX2_RX[7:0] RX2_RXN[7:0]	FlexIO Receive Channel, byte 2
RX2_RXCLK RX2_RXCLKN	FlexIO Receive Channel Clock for byte 2
RX1_RX[7:0] RX1_RXN[7:0]	FlexIO Receive Channel, byte 1
RX1_RXCLK RX1_RXCLKN	FlexIO Receive Channel Clock for byte 1
RX0_RX[7:0] RX0_RXN[7:0]	FlexIO Receive Channel, byte 0
RX0_RXCLK RX0_RXCLKN	FlexIO Receive Channel Clock for byte 0
TX6_TX[7:0] TX6_TXN[7:0]	FlexIO Transmit Channel, byte 6
TX6_TXCLK TX6_TXCLKN	FlexIO Transmit Channel Clock for byte 6
TX5_TX[7:0] TX5_TXN[7:0]	FlexIO Transmit Channel, byte 5
TX5_TXCLK TX5_TXCLKN	FlexIO Transmit Channel Clock for byte 5
TX4_TX[7:0] TX4_TXN[7:0]	FlexIO Transmit Channel, byte 4
TX4_TXCLK TX4_TXCLKN	FlexIO Transmit Channel Clock for byte 4
TX3_TX[7:0] TX3_TXN[7:0]	FlexIO Transmit Channel, byte 3
TX3_TXCLK TX3_TXCLKN	FlexIO Transmit Channel Clock for byte 3
TX2_TX[7:0] TX2_TXN[7:0]	FlexIO Transmit Channel, byte 2
TX2_TXCLK TX2_TXCLKN	FlexIO Transmit Channel Clock for byte 2
TX1_TX[7:0] TX1_TXN[7:0]	FlexIO Transmit Channel, byte 1

Table 5-1. FlexIO Interface Signals (Page 2 of 2)

Signal Name	Description
TX1_TXCLK TX1_TXCLKN	FlexIO Transmit Channel Clock, byte 1
TX0_TX[7:0] TX0_TXN[7:0]	FlexIO Transmit Channel, byte 0
TX0_TXCLK TX0_TXCLKN	FlexIO Transmit Channel Clock, byte 0
RC_REFCLK RC_REFCLKN	FlexIO Reference Clock. Differential input clock. Driven by a 500 MHz clock generated by the Rambus XDR Clock Generator (XCG) module in the system.

5.3.2 FlexIO Power Supplies and References

Table 5-2 lists the FlexIO power supply and reference pins.

Table 5-2. FlexIO Power Supply and Reference Pins (Page 1 of 2)

Pin Name	Description
RC_VDDIO	FlexIO I/O Voltage Supply, 1.20 V. Voltage level and tolerance must conform to the specification in the *Cell Broadband Engine Datasheet*.
RC_VOLREF[1:0]	FlexIO Output Low Voltage Reference. These reference pins are connected on the card to a 56.2/137 ohm divider between RC_VDDIO and RC_VOLGND, which produces a reference voltage of 0.85 V that tracks the RC_VDDIO supply. This voltage calibrates the low swing voltage of the FlexIO channel. FlexIO channel voltage swing is between 1.20 V and 0.850 V. In the *Cell Broadband Engine Datasheet* the relationship between RC_VOLREF and the voltage levels on the interface is specified by means of the Input Common Mode Voltage (V_{ICM}) and Input Voltage Swing (V_{ISW}) parameters.
RC_VOLGND[1:0]	FlexIO Output Low Voltage Reference Ground. These reference pins are dedicated grounds for the RC_VOLREF voltage divider.
RC_ROLREF[1:0]	FlexIO Termination Voltage Reference. These reference pins must be tied to RC_VDDIO in the system.

Table 5-2. FlexIO Power Supply and Reference Pins (Page 2 of 2)

Pin Name	Description
RC_RLOAD[1:0]	FlexIO RLoad Reference. In the system these pins must be tied to RC_VDDIO through a 50 ohm resistor.
RX4_VDDA RX3_VDDA RX2_VDDA RX1_VDDA RX0_VDDA TX6_VDDA TX5_VDDA TX4_VDDA TX3_VDDA TX2_VDDA TX1_VDDA TX0_VDDA	FlexIO Analog Voltage Supply, 1.50 V. Voltage level and tolerance must conform to the specification in the *Cell Broadband Engine Datasheet*.
RX4_GNDA RX3_GNDA RX2_GNDA RX1_GNDA RX0_GNDA TX6_GNDA TX5_GNDA TX4_GNDA TX3_GNDA TX2_GNDA TX1_GNDA TX0_GNDA	FlexIO Analog Voltage Supply Ground.

5.3.3 XDR Memory Interface: Channel 0

The XDR I/O (XIO) interface, formerly known as Yellowstone (YRAC), provides two 32-bit (36-bit if ECC is configured) XDR DRAM channels. Each channel consists of a 32-bit (36-bit if ECC is configured) bidirectional Differential Rambus Signaling Level (DRSL) data interface to external XDR memory devices, a 12-bit unidirectional single-ended Rambus Signaling Level (RSL) command and address bus, and a 4-pin serial interface to the XDR devices that is used for initialization. The data bus (DQ) operates at 3.2 Gbps per pin-pair. The command and address (RQ) bus operates at 800 Mbps per pin, with a 400 MHz reference clock input. The DQ consists of point-to-point signals and the RQ bus is multi-drop.

In a CBE system, the XDR memory channel connects to a set of XDR DRAM Memory devices. The card wiring for the channel must comply with the guidelines in the Rambus XDR System Design Guide. A typical memory configuration connects four 8-bit XDR DRAMs to the XDR memory channel. Memory capacity can be increased by using eight 4-bit XDR DRAMs, or decreased by using two 16-bit XDR DRAMs. All memory configurations provide 12.8 GBps for memory data bandwidth per channel, using a 400 MHz reference-clock input. The CBE processor can be configured to use one or two memory channels.

Table 5-3 on page 138 lists the XDR memory interface signals for channel 0.

Table 5-3. XDR Memory Interface Signals: Channel 0

Signal Name	Description
Y0_DQ3[8:0] Y0_DQ3N[8:0]	XDR DRAM Data Byte 3. Bit 8 is ECC bit 3 when the CBE processor is configured for ECC mode. When not configured for ECC, bit 8 can be left unconnected.
Y0_DQ2[8:0] Y0_DQ2N[8:0]	XDR DRAM Data Byte 2. Bit 8 is ECC bit 2 when the CBE processor is configured for ECC mode. When not configured for ECC, bit 8 can be left unconnected.
Y0_DQ1[8:0] Y0_DQ1N[8:0]	XDR DRAM Data Byte 1. Bit 8 is ECC bit 1 when the CBE processor is configured for ECC mode. When not configured for ECC, bit 8 can be left unconnected.
Y0_DQ0[8:0] Y0_DQ0N[8:0]	XDR DRAM Data Byte 0. Bit 8 is ECC bit 0 when the CBE processor is configured for ECC mode. When not configured for ECC, bit 8 can be left unconnected.
Y0_RQ[11:0]	XIO Request Bus. Provides commands and addresses to the XDR DRAMs.
Y0_RQ_CTM Y0_RQ_CTMN	Differential XIO Clock-To-Master. Reference clock for XDR Memory Interface channel 1. This is a 400 MHz differential clock that is generated in the system by an XDR Clock Generator module. With a 400 MHz reference clock the data rate on the XDR memory interface is 3.2 Gbps per differential pair. It is also possible to run the XDR Memory interface at a slower 3.0 Gbps using a 375 MHz reference clock[1]. If the memory interface is slowed down, the processor clock must also be slowed down by an equivalent amount to avoid buffer underrun in the memory controller. See the *Cell Broadband Engine Datasheet* for clocking requirements.
Y0_RQ_CFM Y0_RQ_CFMN	Differential XIO Clock-From-Master. This is a copy of the Y0_RQ_CTM and Y0_RQ_CTMN clock signals that are routed through the module and back out to the XDR DRAMs.

1. This is a deviation from the Rambus XIO datasheet.

5.3.4 XDR Memory Serial Interface: Channel 0

Table 5-4 lists the XDR memory serial interface signals for channel 0.

Table 5-4. XDR Memory Serial Interface Signals: Channel 0

Signal Name	Description
Y0_RQ_RST	Reset to XDR DRAMs. Active-low output. XDR Reset is asserted during the XDR Initialization portion of the POR sequence to reset the XDR DRAM devices. Its value is controlled by XIO RQ_SERIAL_CTL[0].
Y0_RQ_SCK	Serial Clock to XDR DRAMs. Active-low output. XDR Serial Clock is the strobe used to sample Y0_RQ_RST, Y0_RQ_CMD, and Y0_RQ_SRD. Its value is controlled by XIO RQ_SERIAL_CTL[1].
Y0_RQ_CMD	Serial Command to XDR DRAMs. Active-low output. XDR Serial Command is the serial data from the CBE processor to the XDR memory devices. It is used during the XDR initialization portion of the POR sequence to initialize registers and data within the DRAM. Its value is controlled by XIO RQ_SERIAL_CTL[2].
Y0_RQ_SRD	Serial Read Data from XDR DRAMs. Active-low input. It is used during the XDR initialization portion of the POR sequence to read register data within the DRAM. Its value is sampled by XIO RQ_SERIAL_CTL[3].

5.3.5 Memory Interface (XIO) Power Supplies and References: Channel 0

Table 5-5 lists the memory interface (XIO) power supply and reference pins for channel 0.

Table 5-5. Memory Interface (XIO) Power Supply and Reference Pins: Channel 0

Pin Name	Description
YC_VDDIO	XIO I/O Voltage Supply (1.20V). Voltage level and tolerance must conform to specification in the *Cell Broadband Engine Datasheet*.
Y0_DQ0_VREF Y0_DQ2_VREF	XIO Voltage Reference for testing. Must be tied to YC_VDDIO for normal system operation.
Y0_DQ0_RLOAD Y0_DQ2_RLOAD	XIO RLoad Reference. Must be tied to YC_VDDIO through a 50 ohm resistor.
Y0_RQ_VREF	XIO RQ Reference Voltage. These reference pins must be connected to 39.2/64.9 ohm voltage dividers between YC_VDDIO and GND, which produces a reference voltage of 0.750 V that tracks the YC_VDDIO supply.
Y0_DQC_VOLREF	XIO DQ Reference Voltage. This reference pins must be connected to 50/191 ohm voltage divider between YC_VDDIO and YC_VOLGND, which produces a reference voltage of 0.800 V that tracks the YC_VDDIO supply.
Y0_DQC_VOLGND	XIO DQ Reference Voltage ground. Ground reference for generation of Y0_DQC_VOLREF and Y1_DQC_VOLREF respectively.
Y0_DQC_ROLREF	XIO DQ Resistor Reference. Must be tied to YC_VDDIO through a 50 ohm resister.
Y0_DQ3_VDDA Y0_DQ2_VDDA Y0_DQ1_VDDA Y0_DQ0_VDDA	XIO Analog Voltage Supply, 1.50 V. Voltage level and tolerance must conform to the specification in the *Cell Broadband Engine Datasheet*.
Y0_RQ_VDDA	XIO Analog Voltage Supply, 1.50 V. Voltage level and tolerance must conform to the specification in the *Cell Broadband Engine Datasheet*.
Y0_DQ3_GNDA Y0_DQ2_GNDA Y0_DQ1_GNDA Y0_DQ0_GNDA	XIO Analog Voltage Supply Ground.
Y0_RQ_GNDA	XIO Analog Voltage Supply Ground.

5.3.6 XDR Memory Interface: Channel 1

Table 5-6 lists the XDR memory interface signals for channel 1.

Table 5-6. XDR Memory Interface Signals: Channel 1

Signal Name	Description
Y1_DQ3[8:0] Y1_DQ3N[8:0]	The XDR Memory Interface Channel 1 signals are the same as those for channel 0. The list of signals is included here. For the signal descriptions, see the corresponding signals for channel 0.
Y1_DQ2[8:0] Y1_DQ2N[8:0]	
Y1_DQ1[8:0] Y1_DQ1N[8:0]	
Y1_DQ0[8:0] Y1_DQ0N[8:0]	
Y1_RQ[11:0]	
Y1_RQ_CTM Y1_RQ_CTMN	
Y1_RQ_CFM Y1_RQ_CFMN	

5.3.7 XDR Memory Serial Interface: Channel 1

Table 5-7 lists the XDR memory serial interface signals for channel 1.

Table 5-7. XDR Memory Serial Interface Signals: Channel 1

Signal Name	Description
Y1_RQ_RST	The XDR Memory Serial Interface Channel 1 signals are the same as those for channel 0. The list of signals is included here. For the signal descriptions, see the corresponding signals for channel 0.
Y1_RQ_SCK	
Y1_RQ_CMD	
Y1_RQ_SRD	

5.3.8 XDR Memory Interface (XIO) Power Supplies and References: Channel 1

Table 5-8 lists the XDR memory interface (XIO) power supply and reference pin for channel 1.

Table 5-8. XDR Memory Interface (XIO) Power Supply and Reference Pins: Channel 1

Pin Name	Description
Y1_DQ0_VREF Y1_DQ2_VREF	The XDR memory interface power supplies and references for channel 1 signals are the same as those for channel 0. The list of signals is included here. For the signal descriptions, see the corresponding signals for channel 0.
Y1_DQ0_RLOAD Y1_DQ2_RLOAD	
Y1_RQ_VREF	
Y1_DQC_VOLREF	
Y1_DQC_VOLGND	
Y1_DQC_ROLREF	
Y1_DQ3_VDDA Y1_DQ2_VDDA Y1_DQ1_VDDA Y1_DQ0_VDDA	
Y1_RQ_VDDA	
Y1_DQ3_GNDA Y1_DQ2_GNDA Y1_DQ1_GNDA Y1_DQ0_GNDA	
Y1_RQ_GNDA	

5.3.9 Serial Peripheral Interface (SPI)

The 4-pin serial interface (3 pins plus enable) is compatible with the Serial Peripheral Interface (SPI) protocol. During normal system operation, the CBE processor is a slave device on the SPI interface. An external device operates as the SPI master during the POR sequence to initialize internal CBE registers and calibrate the FlexIO interface (see *Section 2* on page 27 for details). The SPI interface may also be used during CBE-processor operation to monitor CBE-processor status (for example, performance monitor, temperature, recoverable errors, and so forth).

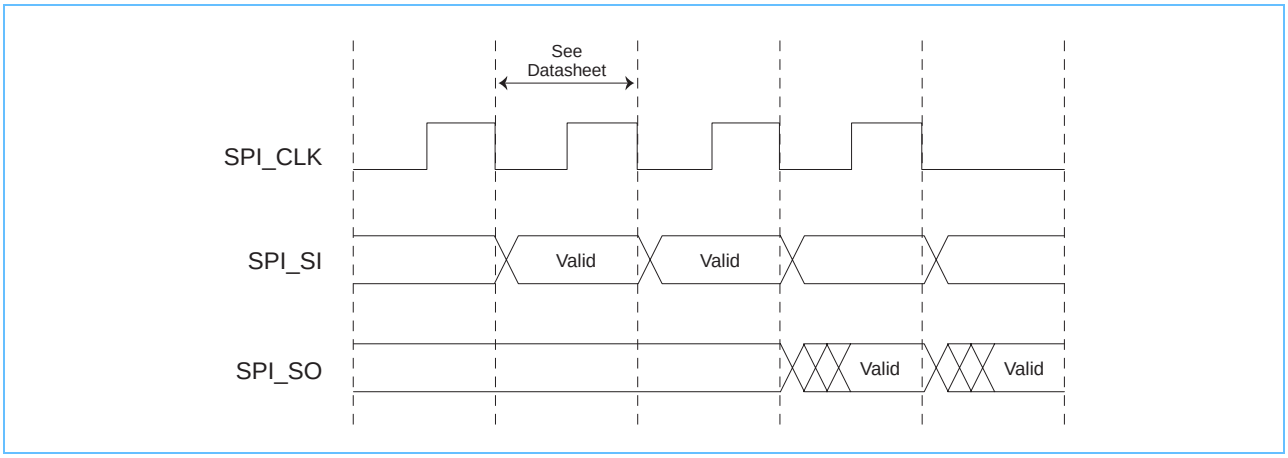
Figure 5-2 on page 142 shows the timing relationship between the serial interface signals. See *Section 3 Serial Peripheral Interface* on page 91 for details on the command and data protocol used on the SPI interface and for definitions of the CBE processor's SPI registers. MMIO registers within the Rambus logic are accessed indirectly through command and data SPI registers in the CBE processor during the POR initialization sequence.

Table 5-9 lists the serial peripheral interface signals.

Table 5-9. Serial Peripheral Interface (SPI) Signals

Signal Name	Description
SPI_CLK	SPI Clock. Active-high input. The rising edge of SPI_CLK captures data from the SPI_SI input and the $\overline{\text{SPI\_EN}}$ input. Output data is driven on SPI_SO on the falling edge of SPI_CLK. Maximum frequency is limited by the CBE Core Clock (NClk) frequency. The maximum CPI_CLK frequency is NClk/20 MHz. For a CBE Core Clock (NClk) frequency of 3.2 GHz, the SPI_CLK must be 160 MHz or slower.
$\overline{\text{SPI\_EN}}$	Serial Peripheral Interface (SPI) Enable. Active-low input. SPI Enable is sampled on the rising edge of SPI_CLK. When asserted, $\overline{\text{SPI\_EN}}$ signals the start of an SPI command sequence and the CBE processor monitors the SPI_SI input for commands addressed to this CBE processor. After the command sequence is complete, $\overline{\text{SPI\_EN}}$ is deasserted to indicate the end. When not asserted, data on SPI_SI is ignored.
SPI_SI	SPI Scan Input. Active-high input. Input data which is sampled on the rising edge of SPI_CLK. The CBE processor examines the incoming command address and processes the command if addressed to this CBE processor.
SPI_SO	SPI Scan Output. Active-high output. Output data in response to commands received on SPI_SI, driven on the falling edge of SPI_CLK. See *Figure 5-2*
SPI_CTL[0:1]	Serial Peripheral Interface (SPI) Control. Active-high input. The SPI control pins are used to configure the CBE-processor number in a multi-CBE-processor system. For a single-CBE-processor system these pins should be tied to ground (multi-chip-ID = 0). In a multi-CBE-processor system, the SPI control pins should be configured as follows: 00 Chip 0 01 Chip 1 10 Chip 2 11 Chip 3 When receiving an SPI command, the CBE processor compares the chip ID to ‘0001’ and the multi-chip ID (bits 4:5 in the SPI Command Field) to the setting of the SPI Control pins. Commands that match the chip ID and multi-chip ID are handled in that CBE processor. Commands that do not match are ignored.

Figure 5-2. SPI Clock and Data Timing



5.3.10 Core PLL

Table 5-10 lists the core PLL pins.

Table 5-10. Core PLL Pins

Pin Name	Description
PLL_REFCLK PLL_REFCLK	Phase Lock Loop Reference clock. Differential input. Reference clock input to the core PLL. Typically this will be a 400 MHz differential clock generated by the XDR Clock Generator. The PLL multiplies this reference clock by eight to produce the very low jitter 3.2 GHz clock (NClk) which is distributed on the internal clock grid.
PLL_BGR	Phase Lock Loop Band-Gap Reference Voltage. Must be left unconnected in system environment. Only used during manufacturing test and debug.
PLL_VDDA	Phase Lock Loop Analog Voltage Supply. PLL_GNDA- Phase Lock Loop Analog Voltage Supply Ground.

5.3.11 Miscellaneous I/O Signals

Table 5-11 lists the miscellaneous I/O signals.

Table 5-11. Miscellaneous I/O Signals (Page 1 of 2)

Signal Name	Description
ATTENTION	Attention. Active-high output. Asserted during system operation to indicate an error condition. During initialization in some system configurations, it is used to request an operation by the system controller. Attention will be asserted asynchronous to other external CBE-processor signals and will remain asserted until software resets the condition that caused the attention.
$\overline{\text{CHECKSTOP\_IN}}$	Checkstop Input. Active-low input. Can be asserted by another device to trigger a Checkstop condition within the CBE processor. Must be tied high (negated) if not used.
$\overline{\text{CHECKSTOP\_OUT}}$	Checkstop Output. Active-low output. Asserted by the CBE processor to indicate an unrecoverable error condition. Should be left unconnected if not used. Checkstop will be asserted asynchronous to other external CBE-processor signals and will remain asserted until software resets the condition that caused the checkstop.
$\overline{\text{HARD\_RESET}}$	$\overline{\text{HARD\_RESET}}$. Hardware reset. Active-low input. Asserted during POR sequence to reset the CBE processor. Negated during normal system operation.
POWER_GOOD	Power good. Active-high input. Negated at the start of the POR sequence to indicate that the power supplies are not yet within specification. Asserted during POR sequence to indicate that power supplies are within specification and stable. Remains asserted during normal system operation.
TBEN	Time Base Enable. Active-high input. Asserted to enable the CBE Time Base function when the CBE processor is configured to use an internal time base. When the CBE processor is configured to use an external time base, the time base clock is provided on this input and can be from 20 MHz to 286 MHz. Actual external time base maximum frequency is dependent on the CBE PLL reference frequency and the CBE-processor configuration. See the Cell Broadband Engine Handbook for equations to calculate this maximum frequency for a specific system configuration.
THERMAL_OVERLOAD	Thermal Overload. Active-high output. Asserted by the CBE processor to indicate that one or more of the on-die thermal sensors has exceeded the configured temperature limit. Thermal Overload will be asserted asynchronous to other external CBE-processor signals and will remain asserted until the thermal overload condition no longer exists.

Table 5-11. Miscellaneous I/O Signals (Page 2 of 2)

Signal Name	Description
VDDS1	Core VDD Voltage Sense. Connected to chip power (VDD) network through a separate signal wire in the module. Due to low current in this sense lead, this BGA lead accurately reflects the voltage level at the C4s. Used in system as feedback voltage for VDD supply, so essentially, VDD is regulated at the chip C4 leads.
VDDS2	Core VDD Voltage Sense Ground. Connected to chip ground network through a separate signal wire in the module. See VDDS1 for details. Due to low current in this sense lead, this BGA lead accurately reflects the ground level at the C4s.
STI_THERMAL[0]	Thermal diode anode.
STI_THERMAL[1]	Thermal diode cathode.
THERMAL_SENSE_POWER	Thermal Sense Power. Analog VDD (1.50 V) supply for triple-point thermal sensor.
THERMAL_SENSE_GND	Thermal Sense Ground. Analog supply ground for triple-point thermal sensor.
THERMAL_SENSE_TEST	Thermal Sense Test. Active-high output. Not used in the normal system environment and should be left unconnected. Used for debug and manufacturing test for the triple-point thermal sensor.

5.3.12 Miscellaneous Test I/O

Table 5-12 lists the miscellaneous test I/O signals.

Table 5-12. Miscellaneous Test I/O Signals (Page 1 of 2)

Signal Name	Description
Reserved_BA19	Tie to ground.
Reserved_AY19	Tie to ground.
Reserved_AW23	Do not connect (leave open).
Reserved_AW18 Reserved_AV18	Tie to ground through 100-ohm resistor.
Reserved_AR22	Tie to ground; however, provide the ability to pull to active-high.
SYS_CONFIG[0:3]	System Configuration. Active-high input. Tied to zero (ground) for normal system operation. These pins are used in manufacturing test and debug to control the test and debug modes as well as the CBE processor's POR sequencer.
Reserved_AV19	Tie to ground.
TRIGGER_IN	Trace Array Trigger Input. Active-high input. Tied to zero (ground) for normal system operation. Not used in normal system operation. For debug, this input can be configured to trigger the capture of data in the on chip logic trace array. The trigger level (active high or low) can be configured to be inverted by means of configuration registers.
TRIGGER_OUT	External Trigger Output. Active-high output. Not used in normal system operation. For debug, this output can be configured to signal that an internal trigger condition has been detected by the logic trace array. The TRIGGER_IN signal can be configured to 'OR' on to the internal trigger signal that is signaled on this output. The output level (active high/low) can be configured to be inverted by means of configuration registers.
Reserved_AW24 Reserved_AV24	Do not connect (leave open).

Table 5-12. Miscellaneous Test I/O Signals (Page 2 of 2)

Signal Name	Description
Reserved_AR21 Reserved_AP21	Do not connect (leave open).
Reserved_AT21	Tie to ground.
Reserved_AV23	Do not connect (leave open).

5.3.13 Power Supply

Table 5-13 lists the power supply pins.

Table 5-13. Power Supply Pins

Pin Name	Description
VDD	Core Voltage Supply. This is the main voltage supply for the internal CBE digital logic. Voltage level and tolerance must conform to the specification in the *Cell Broadband Engine Datasheet*. The nominal voltage for the core voltage supply is determined during manufacturing using a power-performance measurement test and burned into the CBE Voltage Identification (VID) e-fuses. During the POR sequence, this VID value is read out (using the rd_VID SPI register) and used to program the voltage level of the VDD regulator.
GND	Core Voltage Supply Ground.
MC2_VDDIO	Miscellaneous I/O Group Two Voltage Supply (1.20V). Voltage level and tolerance must conform to specifications in the *Cell Broadband Engine Datasheet*. See the *Cell Broadband Engine Datasheet* for details on which miscellaneous I/O pins are powered by MC2_VDDIO.

Appendix A. Memory-Mapped I/O Registers

This section defines the memory map for the CBE processor's Memory-Mapped I/O (MMIO) registers. Register definitions for these registers are given in the *Cell Broadband Engine Registers* document.

Although the Cell Broadband Engine Architecture defines one base register (BP_Base) for relocating the MMIO registers, the CBE processor implements this as several registers, called BE_MMIO_Base, that replicate this relocation function for specific units, as shown in *Table A-1*. These base-register values are initialized from the Configuration Ring during the POR sequence.

The number of bits in these Configuration-Ring fields are also shown in *Table A-1*. In all cases, these bits correspond to the most-significant of the 42-bit real address implemented in the CBE processor. The most-significant 19 bits of all these Configuration-Ring fields should be set to the same value. If a Configuration-Ring field has more that 19 bits, these additional bits should be set to a value consistent with the settings in *Table A-3* on page 148 for the starting address of that unit. Each SPE MFC unit has its own BE_MMIO_Base parameter in the Configuration Ring, but all such BE_MMIO_Base parameters should be initialized to the same value. The IOC unit contains one Configuration-Ring field that defines the most-significant 22 bits of the MMIO space for multiple units, as shown in *Table A-1*.

The value of BE_MMIO_Base is relocatable and the value of the most-significant 19 bits is not specified in this document.

Table A-1. Registers That Are Replicated Forms of BE_MMIO_Base

Configuration-Ring Field	Sets BE_MMIO_Base For
SPE0:7 BE_MMIO_Base Address (19 bits)	SPE0:7
PPE BE_MMIO_Base Address (30 bits)	PPE
MIC BE_MMIO_Base Address (30 bits)	MIC
PRV BE_MMIO_Base Address (30 bits)	Pervasive
BEI BE_MMIO_Base Address (22 bits)	IIC, IOC Address Translation, IOC,BIC, and EIB

A.1 Classification of Registers

The PowerPC Architecture supports three privilege states, which are controlled by MSR[HV] and MSR[PR] bits. MMIO registers are classified according to these states. The states are designed for the following uses:

- *Hypervisor State*—MSR[HV] = '1' and MSR[PR] = '0'. The most-trusted state, used by the hypervisor. Access to all system facilities is provided at this level of privilege.
- *Privileged State*—MSR[HV] = '0' and MSR[PR] = '0'. Used by the operating system within a logical partition.
- *Problem State*—MSR[HV] = '0' and MSR[PR] = '1', or MSR[HV] = '1' and MSR[PR] = '1'. The least-trusted state, used by application software in a logical partition.

If no hypervisor exists on the CBE processor, the entire system is normally run at MSR[HV] = '1', and only two privilege states exist, PR = '0' for firmware and operating-system privileged state, and PR = '1' for application-software problem state.

If address-translation is enabled, privileged software can control, by means of page-table entries, whether application software is given access to particular problem-state MMIO registers; access to the MMIO registers in this mode is not directly enforced by hardware.

A.2 MMIO-Access Rules for 32- and 64-bit Registers

In general, 32-bit registers must be accessed 32 bits at a time, and 64-bit registers must be accessed either 32 bits or 64 bits at a time. No accesses are allowed on less than 32 bits. *Table A-2* lists the access rules for 64-bit registers.

Table A-2. Access Rules for 64-bit Registers

Address Space		Read			Write		
		Doubleword bits 0:63	High Word bits 0:31	Low Word bits 32:63	Doubleword bits 0:63	High Word bits 0:31	Low Word bits 32:63
Problem Space		Allowed	Allowed	Allowed	Allowed	Allowed	Allowed
Privileged Space	High word reserved. Low word defined.	Allowed	Not Allowed	Allowed	Allowed	Not Allowed	Allowed
	High word defined. Low word reserved.	Allowed	Allowed	Not Allowed	Allowed	Allowed	Not Allowed
	High word defined. Low word defined.	Allowed	Not Allowed	Not Allowed	Allowed	Not Allowed	Not Allowed

A.3 MMIO Memory Map

Table A-3 lists the areas of memory that are reserved for MMIO registers.

Bit fields within registers that are marked as reserved are not implemented; writes have no effect and reads return all ones. Reserved areas of the MMIO memory map that are not assigned to a specific functional unit should not be read from or written to. Doing so will cause one of the following software errors:

- For SPE reads or writes to unassigned reserved spaces, at least one of the following `MFC_FIR[46,53,56,58,61]` bits will be set and, in most cases, will cause a Checkstop.
- For PPE reads or writes to unassigned reserved spaces, at least one of the `CIU_FIR[7,8]` bits will be set, and a Checkstop will occur.
- For IOC reads or writes to unassigned reserved spaces, the IOC will respond to the IOIF device that sourced the address request with a ERR (error) response. No IOC FIR bits are set.

Table A-3. CBE-Processor Memory Map (Page 1 of 3)

Base Register from Configuration Ring	Offset From Base Register		Area[1]		Size in Hex
	Start	End			
SPE0 BE_MMIO_Base	x‘00 0000’	x‘03 FFFF’	SPE0	Local Store	x‘4 0000’
	x‘04 0000’	x‘05 FFFF’		Problem State	x‘2 0000’
	x‘06 0000’	x‘07 FFFF’		Privilege 2 Area	x‘2 0000’

Table A-3. CBE-Processor Memory Map (Page 2 of 3)

Base Register from Configuration Ring	Offset From Base Register		Area[1]		Size in Hex
	Start	End			
SPE1 BE_MMIO_Base	x‘08 0000’	x‘0B FFFF’	SPE1	Local Store	x‘4 0000’
	x‘0C 0000’	x‘0D FFFF’		Problem State	x‘2 0000’
	x‘0E 0000’	x‘0F FFFF’		Privilege 2 Area	x‘2 0000’
SPE2 BE_MMIO_Base	x‘10 0000’	x‘13 FFFF’	SPE2	Local Store	x‘4 0000’
	x‘14 0000’	x‘15 FFFF’		Problem State	x‘2 0000’
	x‘16 0000’	x‘17 FFFF’		Privilege 2 Area	x‘2 0000’
SPE3 BE_MMIO_Base	x‘18 0000’	x‘1B FFFF’	SPE3	Local Store	x‘4 0000’
	x‘1C 0000’	x‘1D FFFF’		Problem State	x‘2 0000’
	x‘1E 0000’	x‘1F FFFF’		Privilege 2 Area	x‘2 0000’
SPE4 BE_MMIO_Base	x‘20 0000’	x‘23 FFFF’	SPE4	Local Store	x‘4 0000’
	x‘24 0000’	x‘25 FFFF’		Problem State	x‘2 0000’
	x‘26 0000’	x‘27 FFFF’		Privilege 2 Area	x‘2 0000’
SPE5 BE_MMIO_Base	x‘28 0000’	x‘2B FFFF’	SPE5	Local Store	x‘4 0000’
	x‘2C 0000’	x‘2D FFFF’		Problem State	x‘2 0000’
	x‘2E 0000’	x‘2F FFFF’		Privilege 2 Area	x‘2 0000’
SPE6 BE_MMIO_Base	x‘30 0000’	x‘33FFFF	SPE6	Local Store	x‘4 0000’
	x‘34 0000’	x‘35 FFFF’		Problem State	x‘2 0000’
	x‘36 0000’	x‘37 FFFF’		Privilege 2 Area	x‘2 0000’
SPE7 BE_MMIO_Base	x‘38 0000’	x‘3B FFFF’	SPE7	Local Store	x‘4 0000’
	x‘3C 0000’	x‘3D FFFF’		Problem State	x‘2 0000’
	x‘3E 0000’	x‘3F FFFF’		Privilege 2 Area	x‘2 0000’
SPE0 BE_MMIO_Base	x‘40 0000’	x‘40 1FFF’	SPE0	Privilege 1 Area	x‘2000’
SPE1 BE_MMIO_Base	x‘40 2000’	x‘40 3FFF’	SPE1		x‘2000’
SPE2 BE_MMIO_Base	x‘40 4000’	x‘40 5FFF’	SPE2		x‘2000’
SPE3 BE_MMIO_Base	x‘40 6000’	x‘40 7FFF’	SPE3		x‘2000’
SPE4 BE_MMIO_Base	x‘40 8000’	x‘40 9FFF’	SPE4		x‘2000’
SPE5 BE_MMIO_Base	x‘40 A000’	x‘40 BFFF’	SPE5		x‘2000’
SPE6 BE_MMIO_Base	x‘40 C000’	x‘40 DFFF’	SPE6		x‘2000’
SPE7 BE_MMIO_Base	x‘40 E000’	x‘40 FFFF’	SPE7		x‘2000’
	x‘41 1000’	x‘4F FFFF’	*Reserved*		
PPE BE_MMIO_Base	x‘50 0000’	x‘50 0FFF’	PPE	Privilege Area	x‘1000’
	x‘50 1000’	x‘50 7FFF’	*Reserved*		
BEI BE_MMIO_Base	x‘50 8000’	x‘50 8FFF’	IIC		x‘1000’

Table A-3. CBE-Processor Memory Map (Page 3 of 3)

Base Register from Configuration Ring	Offset From Base Register		Area[1]		Size in Hex
	Start	End			
PRV BE_MMIO_Base	x‘50 9000’	x‘50 93FF’	PRV	Trace Logic Array	x‘0400’
	x‘50 9400’	x‘50 97FF’		Performance Monitor	x‘0400’
	x‘50 9800’	x‘50 9BFF’		Thermal and Power Management	x‘0400’
	x‘50 9C00’	x‘50 9FFF’		Reliability, Availability, Serviceability (RAS)	x‘0400’
MIC BE_MMIO_Base	x‘50 A000’	x‘50 AFFF’	MIC and TKM		x‘1000’
	x‘50 B000’	x‘50 FFFF’	*Reserved*		
BEI BE_MMIO_Base	x‘51 0000’	x‘51 0FFF’	IOC	I/O Address Translation	x‘1000’
	x‘51 1000’	x‘51 13FF’	BIC	BIC0 NClk	x‘0400’
	x‘51 1400’	x‘51 17FF’		BIC1 NClk	x‘0400’
	x‘51 1800’	x‘51 1BFF’	EIB		x‘0400’
	x‘51 1C00’	x‘51 1FFF’	IOC	I/O Commands	x‘0400’
	x‘51 2000’	x‘51 2FFF’	BIC	BIC0 BClk	x‘1000’
	x‘51 3000’	x‘51 3FFF’		BIC1 BClk	x‘1000’
	x‘51 4000’	x‘7F FFFF	*Reserved*		
	x‘80 0000’	System Maximum	*Available to Software*		

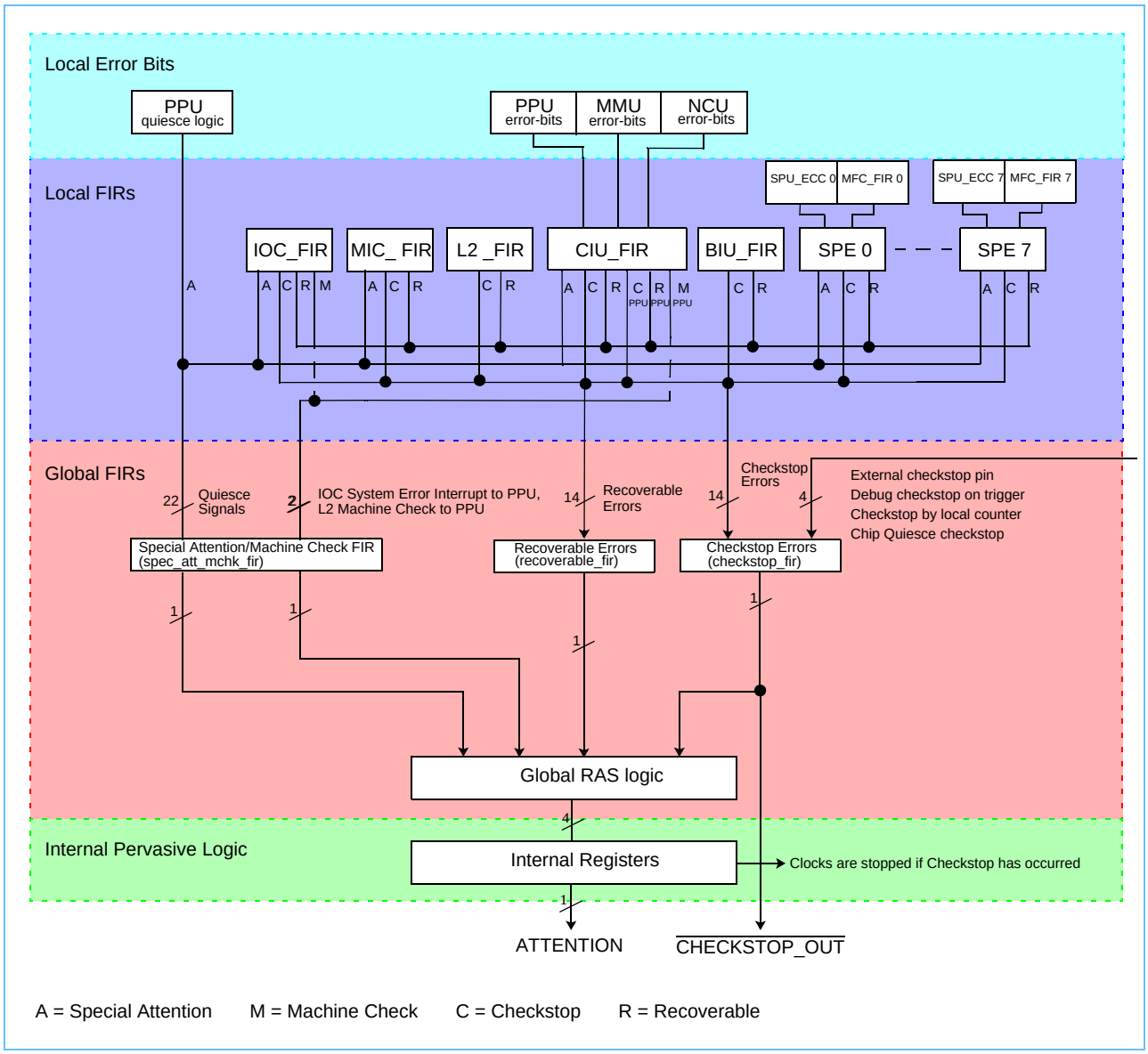
1. See *Section 11.3.1 Access Privilege* on page 348 for definitions of SPE Privilege 1 and Privilege 2.

Appendix B. Fault Isolation Register Overview

The CBE processor contains several Fault Isolation Registers (FIRs) for collecting and reporting information about errors. FIR registers are organized in two categories—Global FIRs and Local FIRs. The lowest level of error-collection is in the Local FIRs, which are implemented in various units on the chip. The errors are then ORed together and reported in the Global FIRs, which are implemented in the Test Control Unit (TCU) within the CBE processor's Pervasive Logic.

Figure B-1 shows the different levels of reporting. All FIRs are initialized to zero at POR. The FIRs do not get initialized to their operational modes and settings until the operating system boots. The Quiesce Logic in *Figure B-1* is shown here for completeness. It is part of a debug mode for the CBE processor and is beyond the scope of this document.

Figure B-1. Error Reporting Structure



During POR, the System Controller will see the ATTENTION signal on the CBE processor activated several times as part of the POR sequence. After POR is complete, the only times the System Controller will see the ATTENTION signal active are for error-related reasons during normal operation. The ATTENTION signal is activated when the CBE processor has a livelock or certain error conditions that have been recorded in the FIR registers and that require the System Controller and the operating system to intervene.

B.1 Local FIRs

All of the local errors are captured in one of the Local FIR MMIO registers, which include:

- `IOC_FIR` (includes error bits for the BEI and EIB).
- `MIC_FIR`.
- `L2_FIR`.
- `CIU_FIR` (includes the error bits for the MMU, NCU, and the rest of the PPU).
- `BIU_FIR`.
- `MFC_FIR`s (eight, one for each SPE).
- `SPU_ECC`s (eight, one for each SPE).

Each Local FIR is implemented as a group of MMIO registers, including:

- FIR (<*unit*>`_FIR`).
- FIR Set (<*unit*>`_FIR_Set`).
- FIR Reset (<*unit*>`_FIR_Reset`).
- FIR Error Mask (<*unit*>`_FIR_Err`).
- FIR Error Mask Set (<*unit*>`_FIR_Err_Set`).
- FIR Error Mask Reset (<*unit*>`_FIR_Err_Reset`).
- FIR Checkstop Enable (<*unit*>`_FIR_ChkStpEnbl`).

See the *Cell Broadband Engine Registers* document for details. The `IOC_FIR` group omits the Error Mask Set and Error Mask Reset registers but includes one additional register, System Error Enable register (`IOC_FIR_SysErrEnbl`), which is used to generate an enable for the System Error Interrupt. Also, the `MIC_FIR` and the `SPU_ECC` registers are implemented differently than the other Local FIRs.

Each Local FIR is implemented with sticky bits. This means that after any bit is set, it remains set until an MMIO write to a FIR Reset register resets the bit. Resetting FIR bits through MMIO writes is only possible if a Checkstop does not occur, because a Checkstop would stop the clocks.

Each error bit in a Local FIR can be configured to be reported as a Checkstop or a Recoverable Error (but not both). This is done by setting the corresponding bit in the FIR Checkstop Enable register to ‘1’ for Checkstop reporting or ‘0’ for Recoverable Error reporting. If configured to report as a Checkstop and enabled with the Global Fault Isolation Error Enable Mask Register (`fir_enable_mask`), the error will cause the ATTENTION and $\overline{\text{CHECKSTOP\_OUT}}$ signals to be asserted. In addition, the Local FIR will be locked to prevent latching any subsequent errors.

The FIR Error Mask Set and FIR Error Mask Reset work exactly the same way as the FIR Set and FIR Reset, except that they operate on the FIR Error Mask register instead of the FIR register.

B.1.1 Local FIR Logic Diagrams

Figure B-2 through *Figure B-4* on page 154 shows the logic behind the bit implementations for various Local FIR registers. The Checkstop, Recoverable, and Block signals are for the entire Local FIR. *Figure B-2* shows the logic per bit for most Local FIR registers (except the SPU ECC, `MIC_FIR`, and `IOC_FIR` registers). *Figure B-3* shows the extra Machine Check logic for `L2_FIR[46]`. *Figure B-4* shows the logic per bit in the `IOC_FIR` register, including the System Error Interrupt.

Figure B-2. Local FIR Logic Diagram Per Bit (General Case)

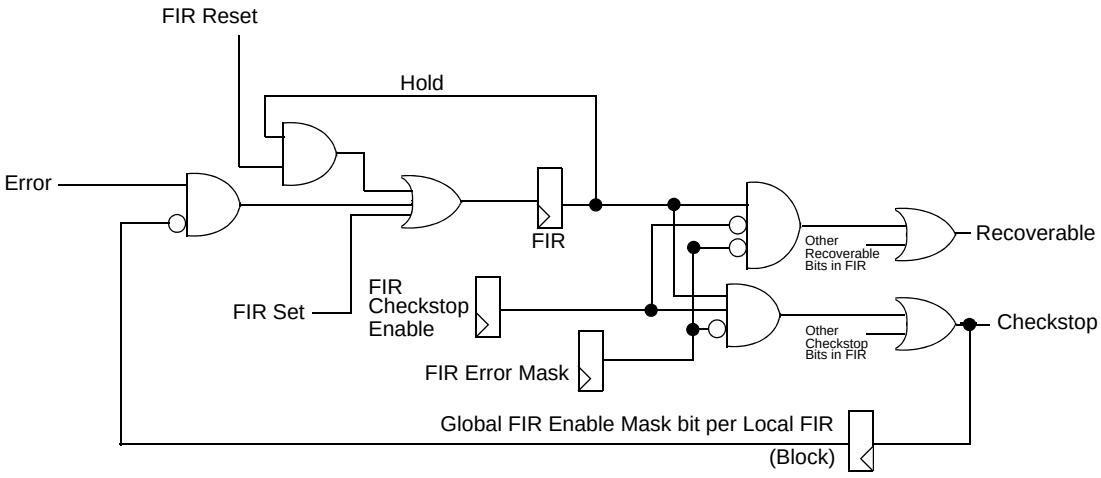


Figure B-3. L2_FIR[46] Logic Diagram—Machine Check to PPU

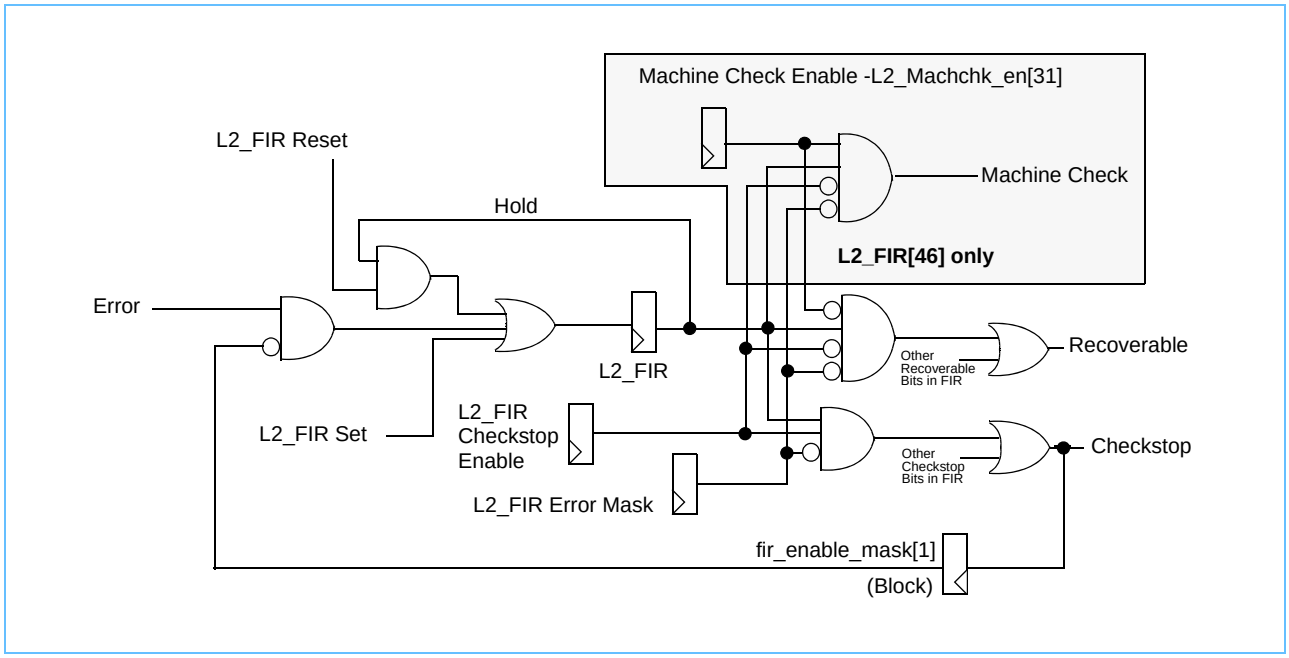
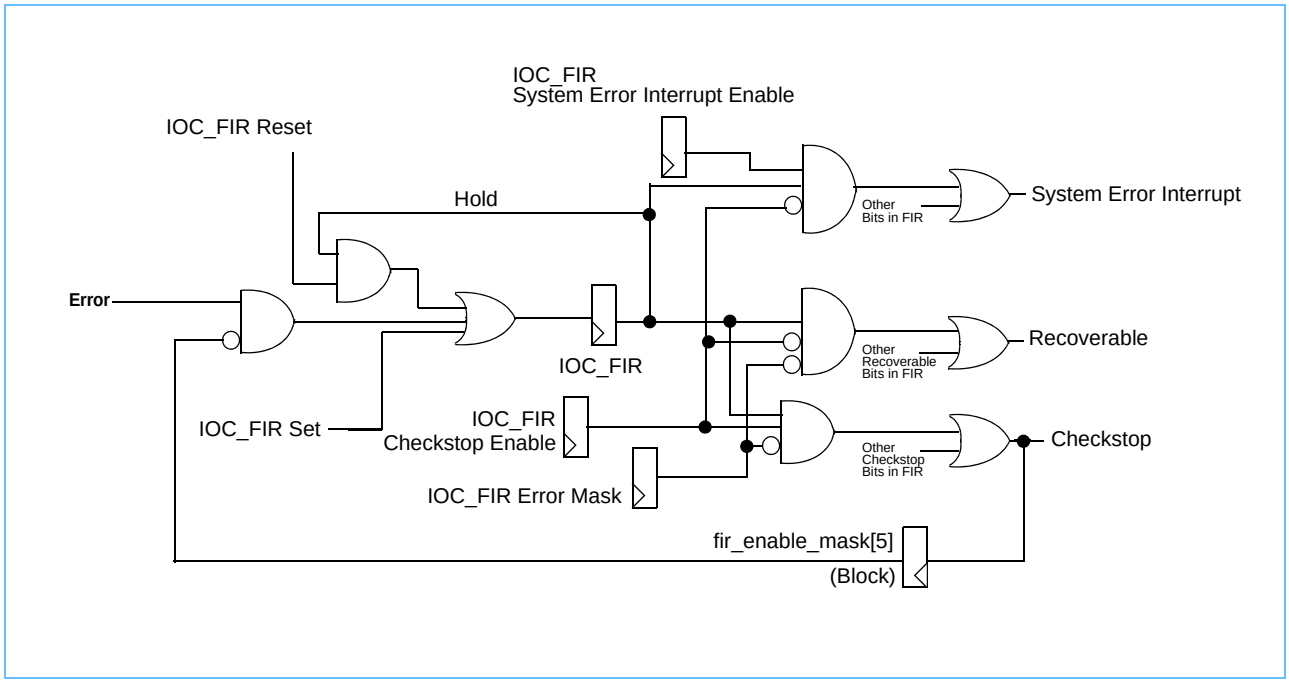


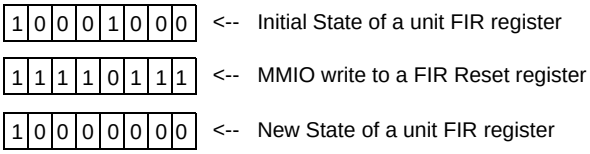
Figure B-4. Local FIR Logic Diagram Per Bit for the IOC_FIR Register



B.1.2 Setting, Resetting, and Masking Errors in Local FIRs

Among the Local FIRs, the FIR Set and Reset registers are also mask registers for the FIR registers. The FIR registers are read-only, whereas the FIR Set and Reset registers are write-only. All of these registers are initialized to zeros during POR. FIR bits are set in the FIR register when an error occurs. They can also be set by using the FIR Set mask register or cleared using the FIR Reset mask register. *Figure B-5* shows an example of a bit that is cleared using the FIR Reset mask register.

Figure B-5. Reset of a Local FIR (General Case)



The FIR Error Mask register gates the effect of the FIR bits being set; a ‘1’ in any bit position means ignore the corresponding bit in the FIR). The FIR Checkstop Enable register is used to select whether or not to generate a Checkstop when a FIR bit is set. The Local FIR mask registers should be set by the operating system after the PPE firmware-initialization sequence. The FIR register collects multiple recoverable errors prior to a Checkstop, but holds the register content after a Checkstop occurs.

The FIR Error Mask Set (<*unit*>`_FIR_Err_Set`) and FIR Error Mask Reset (<*unit*>`_FIR_Err_Reset`) registers work exactly the same way as the FIR Set and FIR Reset, except that they operate on the FIR Error Mask register instead of the FIR register.

B.2 Global FIR registers

In addition to the Local FIRs, the CBE processor contains several Global FIR MMIO registers, including:

- Global Fault Isolation Register (`checkstop_fir`).
- Global Fault Isolation Register For Recoverable Errors (`recoverable_fir`).
- Global Fault Isolation Register For Special Attention And Machine Check (`spec_att_mchk_fir`).
- Global Fault Isolation Mode Register (`fir_mode_reg`).
- Global Fault Isolation Error Enable Mask Register (`fir_enable_mask`).
- Local Error Counter Status Register (`loc_cn_status_reg`).

B.2.1 Global Checkstop FIR

The `checkstop_fir` register contains the unmasked Checkstop status of all of the sources that could cause a Checkstop state (clocks stopped) on the CBE processor. All of these Checkstops can be individually masked with the `fir_enable_mask` register. The `checkstop_fir[28]` bit shows the state of the $\overline{\text{CHECKSTOP\_IN}}$ signal.

All of the error bits configured to report a Checkstop (if the corresponding Checkstop enable bit is set in the FIR Checkstop Enable register) are ORed together to provide the unit Local FIR Checkstop status in the `checkstop_fir` register. Any bit set in the `checkstop_fir` register causes the following responses:

- `checkstop_fir` register freezes.
- `recoverable_fir` register freezes.
- $\overline{\text{CHECKSTOP\_OUT}}$ signal asserted.
- ATTENTION signal asserted.

Note: *The Global Fault Isolation Mode Register (`fir_mode_reg`) has a control bit, Enable Checkstop by SPEs (bit 26) which affects how Checkstops are processed for SPEs. When `fir_mode_reg[26]` is set to '1' the checkstop_fir register is enabled to work correctly for SPE Checkstops. However, when set to '0' (the POR value), SPE Checkstops are handled by an interrupt to the PPE. See Appendix B.2.4 Global FIR Mode on page 156.*

B.2.2 Global Recoverable FIR

The `recoverable_fir` register has a unique bit for each Local FIR which is the OR of the errors configured in the FIR Checkstop Enable (<*unit*>`_FIR_ChkStpEnbl`) register to report as recoverable errors from that Local FIR. The bits in this register reflect the current state of the respective Local FIRs until a Checkstop occurs, in which case the register freezes. The `recoverable_fir` register can also be configured to freeze on the first incoming recoverable error by setting the `fir_mode_reg[19]` bit.

B.2.3 Global FIR Error Enable Mask

The fir_enable_mask register allows the individual masking of Checkstop, Recoverable, Special Attention, and Machine Check errors on the various sources. The initial state of this register is all zeros, which means that all errors are masked. Writing a ‘1’ to any of the register bits enables that error (unmasks the error). This is opposite to the masking method used in the local <*unit*>`_FIR_Err` registers, which mask by writing a ‘1’ and unmask (the POR default) by writing a ‘0’.

B.2.4 Global FIR Mode

The fir_mode_reg control register is used for setting the behavior of the FIRs and various signals used in debug. Only three bits are used in this register for normal operation: `fir_mode_reg[19, 26, 27]`. The remaining bits are used for debug and should be ignored.

Setting `fir_mode_reg[19]` to ‘1’ causes the `recoverable_fir` register to behave like the `checkstop_fir` register, meaning that the first occurrence of a recoverable error freezes the `recoverable_fir` register and blocks further incoming errors.

The `fir_mode_reg[26]` bit is an additional control bit for the behavior of Checkstops coming from any of the SPEs. Leaving `fir_mode_reg[26]` set to ‘0’ (the POR default) causes Checkstops from the SPE to send an external interrupt to the PPE and be handled by means of the interrupt routine rather than hold the clocks and activate the external $\overline{\text{CHECKSTOP\_OUT}}$ signal. Setting

fir_mode_reg[26] to '1' allows Checkstops in the SPEs to behave as Checkstops in other units (hold the clocks and activate the CHECKSTOP_OUT signal). The desired behavior for SPE Checkstops is decided by the operating system.

Setting the fir_mode_reg[27] bit enables overflows in any of the Local Recoverable Error Counter Registers (see *Section B.2.6*), which are recorded in the loc_cn_status_reg register, to cause a Checkstop. This bit is a mask bit similar to the mask bits in the fir_enable_mask register.

B.2.5 Global FIR for Special Attention and Machine Check

The spec_att_mchk_fir status register contains only two bits:

- qo—CBE processor is in a quiesced state (specatt./quiesce).
- m0—PPU (imprecise machine check) or IOC System Error Interrupt.

The Special Attention and Quiesced states (specatt./quiesce) are part of an internal debug facility and are not part of the scope of this document. The m0 bit has two sources—the Machine Check from L2_FIR[46] ORed with the System Error Interrupt from the IOC_FIR. If the L2_FIR[46] bit is '0', the unmasked error from L2_FIR[46] can result in a recoverable error or a Machine Check error. If L2_Machchk_en[63] is '1' and fir_enable_mask[31] is '1', then the error will be recorded as a Machine Check error in spec_attn_mchk_fir[31] and sent to the PPE as a Machine Check.

The System Error interrupt is sent to the PPE and recorded in spec_att_mchk_fir[31] when the following two conditions are met:

- An unmasked error occurs on any of the bits in the IOC_FIR that have their corresponding system error interrupt enable bits set, and
- The fir_enable_mask[31] bit is set to '1'.

B.2.6 Local Recoverable Error Counters and Local Error Counter Status

All recoverable errors for the MIC, CIU, and SPU Local FIRs are counted in the following local recoverable error counter MMIO registers:

- MIC ErrorMask/RecErrorEnable/Debug Control (MIC_FIR_Debug).
- CIU Local Recoverable Error Counter (CIU_REC).
- SPU ECC Status Register (SPU_ECC_Stat).

Overflows to these counters are recorded in the Local Error Counter Status Register (loc_cn_status_reg) and can be set up to cause a Checkstop if the fir_mode_reg[27] bit is set to '1'.

Appendix C. Livelock Resolution Mode

Livelocks occur when one or more units in a processor element cannot make forward progress. The CBE processor contains several internal mechanisms avoid livelock. One example is the pseudo-random retry backoff mechanism. Although this mechanism eliminates most livelocks, some can still occur. In addition to the internal mechanisms, the CBE processor provides an external notification to the System Controller when a livelock is detected and is not resolved. The notification is in the form of an ATTENTION signal to the System Controller. In response to the ATTENTION signal, the System Controller can further alter the operation by enabling the *livelock resolution mode*, which alters the operation of the processor and usually resolves the livelock.

Like livelocks, rare cases exist in which a processing element is making forward progress, but at an extremely slow rate. This is referred to as *starvation*. The internal mechanisms are usually enough to prevent starvation. Starvation is not detected by the CBE processor. However, a System Controller can randomly, and at a very slow rate, enable and then disable the livelock resolution mode to resolve any condition causing starvation.

For performance reasons, the System Controller should never leave the CBE processor in the livelock resolution mode for an extended period of time. The system should provide a mechanism for the System Controller to notify the operating system that a livelock was detected and resolved.

C.1 System Controller Actions

The ATTENTION signal is asserted when the CBE processor detects a livelock condition. This signal is the same as the ATTENTION signal used during the POR sequence. Other conditions can also cause the ATTENTION signal to be asserted. The System Controller should monitor the ATTENTION signal using either polling or interrupts. When the ATTENTION signal is asserted, the System Controller should read the Read SPI Status register (`rd_spi_status`). If `rd_spi_status[0,7]` are both set, the CBE processor has detected a livelock condition. The System Controller should then perform the following sequence:

1. Write `wr_spi_status[18]` = ‘1’ to throttle the PPE.
2. Write `wr_spi_status[16,17,19]` = ‘1’ to quiesce transactions and enable the livelock resolution mode:
   - `wr_spi_status[16]` = ‘1’ quiesces MFC bus transactions.
   - `wr_spi_status[17]` = ‘1’ quiesces PPE bus transactions.
   - `wr_spi_status[19]` = ‘1’ enables livelock resolution mode.
3. Write wr_spi_status[18] = ‘0’ to stop throttling the PPE. (If this step is not performed, the CBE processor will not resolve the livelock.)
4. Write `wr_spi_status[4]` = ‘1’ to reset and resample the livelock condition by deactivating the ATTENTION signal.
5. Read `rd_spi_status[7]` to determine if the livelock is resolved. This bit will return ‘0’ if the livelock is resolved, or it will return ‘1’ if the livelock is not resolved.
6. If the livelock is resolved:
   a. Write `wr_spi_status[16,17,19]` = ‘0’ to remove the quiesce for the MFC and PPE bus transactions, and to disable the livelock resolution mode.

b. Notify the operating system to indicate that a livelock has been detected and resolved.

7. If the livelock is not resolved, then the System Controller should assert the $\overline{\text{CHECKSTOP\_IN}}$ signal and perform any system-dependent operations for reporting a Checkstop condition.

Optionally, the System Controller can perform Step 1 through 6a at random intervals to resolve any potential starvation conditions. Steps 1 through 6 should be performed sequentially, and the interval between performing these steps should be random and low-frequency. The notification to the operating system in Step 6b should not be performed.

C.2 Configuration Ring Settings

In order for a livelock condition to be recorded, the Configuration-Ring settings must be set as follows:

- Bit [2490] L2 Livelock Indication Enable = ‘1’.
- Bit [2602] NCU Livelock Indication Enable = ‘1’.
- Bit [2603] PPE Livelock Indication Enable = ‘1’.
- Bit [1080] AC0 Livelock Response Control = ‘0’
- Bit [1138] AC1 Livelock Response Control = ‘0’
- Bit [2675] Pervasive Livelock Indication Enable = ‘1’.

See *Section 4 Configuration Ring* on page 119 for details about these bits.

C.3 Fault Isolation Bit Settings

To disable a Checkstop, the livelock resolution mode requires the following configuration of FIR bits:

- `CIU_FIR_ChkStpEnbl[3,6,7,8,9,10,11]` = ‘0’.
- `L2_FIR_ChkStpEnbl[49,52]` = ‘0’.
- `IOC_FIR_ChkStpEnbl[42,54]` = ‘0’.

In addition, to allow the System Controller to cause a Checkstop if livelocks cannot be resolved during livelock resolution mode, the following FIR bit must be configured to enable the $\overline{\text{CHECKSTOP\_IN}}$ signal (C4 pin):

- `fir_enable_mask[14]` = ‘1’.

See *Appendix B Fault Isolation Register Overview* on page 151 for details about these bits.

C.4 Operating-System Requirements

When the operating system receives a system-dependent notification from the System Controller, the local FIR registers should be read. If a FIR bit is set, the operating system should reset and record the FIR information and the fact that a livelock was detected and resolved. The following FIR settings should be read, recorded, and reset when a livelock is resolved:

- `MFC_FIR[47,54,57,59,62]` for each SPE.
- `CIU_FIR[3,6,7,8,9,10,11]`.
- `L2_FIR[49,52]`.
- `IOC_FIR[42,54]`.

Appendix D. DQ Syndrome-to-Pin Mapping

For RQ debugging, there is no facility inside the MIC to aid with a command-bus problem. Incorrect commands are simply not received by the XDR DRAMs correctly. Certain bits (such as address bits) may cause address parity errors.

For DQ debugging, (assuming a single-bit error) the DQ syndrome-to-pin mapping is shown in *Table D-1*. From the syndromes, the failing pin will be on one of the four DQ blocks on that memory channel. Therefore, in *Table D-1*, the notation Y<0..1>_DQ<0..3>(*n)* in the DQ Pin column means that the syndrome is pointing to memory channel 0 or 1, DQ block 0, 1, 2, or 3, pin *n*.

Table D-1. DQ Syndrome-to-Pin Mapping (Page 1 of 3)

Internal Data Bit	MIC_Ecc_Addr_*n* [*n* = 0 or 1] Syndrome	DQ Pin
Bit (0)	‘10100100’	Y<0..1>_DQ<0..3>(7)
Bit (1)	‘11000100’	
Bit (2)	‘11000010’	
Bit (3)	‘10100010’	
Bit (4)	‘10011110’	
Bit (5)	‘11000001’	
Bit (6)	‘10100001’	
Bit (7)	‘10010001’	
Bit (8)	‘01010010’	Y<0..1>_DQ<0..3>(6)
Bit (9)	‘01100010’	
Bit (10)	‘01100001’	
Bit (11)	‘01010001’	
Bit (12)	‘01001111’	
Bit (13)	‘11100000’	
Bit (14)	‘11010000’	
Bit (15)	‘11001000’	
Bit (16)	‘00101001’	Y<0..1>_DQ<0..3>(5)
Bit (17)	‘00110001’	
Bit (18)	‘10110000’	
Bit (19)	‘10101000’	
Bit (20)	‘10100111’	
Bit (21)	‘01110000’	
Bit (22)	‘01101000’	
Bit (23)	‘01100100’	

Table D-1. DQ Syndrome-to-Pin Mapping (Page 2 of 3)

Internal Data Bit	MIC_Ecc_Addr_*n* [*n* = 0 or 1] Syndrome	DQ Pin
Bit (24)	‘10010100’	Y<0..1>_DQ<0..3>(4)
Bit (25)	‘10011000’	
Bit (26)	‘01011000’	
Bit (27)	‘01010100’	
Bit (28)	‘11010011’	
Bit (29)	‘00111000’	
Bit (30)	‘00110100’	
Bit (31)	‘00110010’	
Bit (32)	‘01001010’	Y<0..1>_DQ<0..3>(3)
Bit (33)	‘01001100’	
Bit (34)	‘00101100’	
Bit (35)	‘00101010’	
Bit (36)	‘11101001’	
Bit (37)	‘00011100’	
Bit (38)	‘00011010’	
Bit (39)	‘00011001’	
Bit (40)	‘00100101’	Y<0..1>_DQ<0..3>(2)
Bit (41)	‘00100110’	
Bit (42)	‘00010110’	
Bit (43)	‘00010101’	
Bit (44)	‘11110100’	
Bit (45)	‘00001110’	
Bit (46)	‘00001101’	
Bit (47)	‘10001100’	
Bit (48)	‘10010010’	Y<0..1>_DQ<0..3>(1)
Bit (49)	‘00010011’	
Bit (50)	‘00001011’	
Bit (51)	‘10001010’	
Bit (52)	‘01111010’	
Bit (53)	‘00000111’	
Bit (54)	‘10000110’	
Bit (55)	‘01000110’	

Table D-1. DQ Syndrome-to-Pin Mapping (Page 3 of 3)

Internal Data Bit	MIC_Ecc_Addr_*n* [*n* = 0 or 1] Syndrome	DQ Pin
Bit (56)	‘01001001’	Y<0..1>_DQ<0..3>(0)
Bit (57)	‘10001001’	
Bit (58)	‘10000101’	
Bit (59)	‘01000101’	
Bit (60)	‘00111101’	
Bit (61)	‘10000011’	
Bit (62)	‘01000011’	
Bit (63)	‘00100011’	
Bit (64)	‘11000111’	
Bit (65)	‘11111000’	
Bit (66)	‘00011111’	
Bit (67)	‘10000000’	Y<0..1>_DQ<0..3>(8)
Bit (68)	‘01000000’	
Bit (69)	‘00100000’	
Bit (70)	‘00010000’	
Bit (71)	‘00001000’	
Bit (72)	‘00000100’	
Bit (73)	‘00000010’	
Bit (74)	‘00000001’	

Glossary

ABIST	Automatic Built-In Self Test.
AC0	Address Concentrator 0.
ATO	Atomic Unit. Part of an SPE's MFC. It is used to synchronize with other processor units.
b	Bit.
B	Byte.
backing store	Any type of nonvolatile data storage, including disk, diskette, and tape. Compare *main storage*.
BClk	Bus Interface Controller (BIC) Core Clock.
BE	See *Cell Broadband Engine*.
BE Bus	Broadband Engine Bus. An early name (now unused) for the Element Interconnect Bus (EIB).
BED	Cell Broadband Engine Distribution Bus.
BEI	Cell Broadband Engine Interface.
BHT	Branch History Table.
BIC	Bus Interface Controller. Part of the Broadband Engine Interface (BEI) to I/O.
BIF	Broadband Processor Interface, or simply Broadband Interface. The EIB's internal communication protocol. It supports coherent interconnection for to other Cell Broadband Engines and BIF-compliant I/O devices, such as memory subsystems, switches, and bridge chips. See *IOIF*.
BIF/IOIF0	One of two I/O interfaces (also called IOIF0). It is software-selectable between the non-coherent IOIF protocol and the fully coherent Broadband Interface (BIF) protocol—the EIB's native internal protocol—which coherently extends the EIB to another device that can be another CBE processor.
big-endian	An ordering of bytes and bits in which the lowest-address byte and lowest-numbered bit are the most-significant (high) byte and bit, respectively. The Cell Broadband Engine supports only big-endian ordering; it does not support little-endian ordering.
BIU	Bus Interface Unit. Part of the PPE's interface to the EIB.
BMT	Bandwidth Management Table.
BRU	Branch Unit.

caching-inhibited	A memory update policy in which caches are bypassed, and the load or store is performed to or from main storage. Only the storage location specified by the instruction (rather than a full cache line) is accessed at a caching-inhibited location. Stores to caching inhibited pages must update the memory hierarchy to a level that is visible to all processors and devices in the system. The operating system typically implements this policy, for example, for I/O devices.
CBE processor	Cell Broadband Engine processor.
CBEA	Cell Broadband Engine Architecture. The Cell Broadband Engine is one implementation of the Cell Broadband Engine Architecture. Same as *BPA*.
CBE core clock	NClk.
CEC	Central Electronics Complex.
channel	An internal register or queue used for communication between PPE and an SPE. Channels also provide ways for PPE and SPEs to communicate with the EIB. For example, an SPU uses channels to talk to the Mailbox, to synchronize, or to update the decrementers.
chicken switch	A switch that disable optimizing functions, such as caches, so that problems can be isolated for debug.
CIU	Core Interface Unit.
CL	A class-ID parameter in an MFC command.
core clock	The CBE Core Clock (NClk).
corner case	A situation in which a rare combination of factors creates an unusual occurrence or a situation that involves the occurrence of the extreme values or limits of several variables.
CR	Condition Register.
CRC	Cyclic Redundancy Check
CSA	Context-Save Area.
CSRA	Context Save And Restore Area.
CTR	Count Register.
DAR	Data Address Register.
dcbf	Data cache block flush instruction.
dcbst	Data cache block store instruction.
dcbt	Data cache block touch x form instruction.
dcbtst	Data cache block touch for store instruction.

dcbz	Data cache block zero instruction.
D-ERAT	Data ERAT, or the table that contains such translations, or a table entry that contains such a translation.
DERR	Data Error.
DMA	Direct Memory Access. A technique for using a special-purpose controller to generate the source and destination addresses for a memory or I/O transfer.
DMAC	Direct Memory Access Controller. A controller that performs DMA transfers.
DR	Data relocate
DSI	Data Storage Interrupt.
DSISR	Data Storage Interrupt Status Register.
EA	Effective Address.
EAH	An effective address high parameter in an MFC command.
EAL	An effective address low parameter in an MFC command.
ECB	External Control Bus.
ECC	Error-Correcting Code.
effective address	An address generated or used by a program to reference memory. A memory-management unit translates an effective address (EA) to a virtual address (VA), which it then translates to a real address (RA) that accesses real (physical) memory. The maximum size of the effective-address space is 2^{64} bytes.
EIB	Element Interconnect Bus. The on-chip coherent bus that handles communication between the PPE, SPEs, memory, and I/O devices (or a second Cell Broadband Engine). The EIB is organized as four unidirectional data rings (two clockwise and two counterclockwise).
EIC	External Interrupt Controller.
eieio	Enforce in-order execution of I/O transaction, a PowerPC instruction.
ERAT	Effective-to-Real Address Translation, or a buffer or table that contains such translations, or a table entry that contains such a translation.
ESID	Effective segment ID
exception	An error, unusual condition, or external signal that may alter a status bit and will cause a corresponding interrupt, if the interrupt is enabled. See *interrupt*.

fence	An option for a barrier ordering command that causes the processor to wait for completion of all MFC commands before starting any commands queued after the **fence** command. It does not apply to these immediate commands: **getllar**, **putllc** and **putlluc**.
FIR	Fault Isolation Registers. These are part of the RAS unit.
FlexIO	Rambus FlexIO bus.The physical-link I/O signals on the BIF and IOIF interfaces.
FLIH	First-Level Interrupt Handler.
FP	Floating Point.
FPR	Floating-Point Register.
FPU	Floating-Point Unit.
FXU	Fixed-point exception unit
GPR	General-Purpose Register.
GRF	Growable Array File.
HDEC	Hypervisor Decrementer.
HID	Hardware-Implementation Dependent.
hypervisor	A control (or virtualization) layer between hardware and the operating system. It allocates resources, reserves resources, and protects resources among (for example) sets of SPEs that may be running under different operating systems. The CBE processor has three operating modes: user, supervisor and hypervisor. The hypervisor performs a meta-supervisor role that allows multiple independent supervisors' software to run on the same hardware platform. For example, the hypervisor allows both a real-time operating system and a traditional operating system to run on a single PPE. The PPE can then operate a subset of the SPEs in the CBE processor with the real-time operating system, while the other SPEs run under the traditional operating system.
IABR	Instruction Address Breakpoint Register.
IBUF	Instruction Dispatch Buffer.
IC	Instruction Cache.
ICB	Internal Configuration Bus.
ID	Instruction Dispatch.

I-ERAT	Instruction ERAT, or the table that contains such translations, or a table entry that contains such a translation.
IFAR	Instruction Fetch Address Register.
IIC	Internal Interrupt Controller.
INT	Interrupt.
interrupt packet	Used to signal an interrupt, typically to a processor or to another interruptible device.
IOC	I/O Interface Controller.
I/O device	Input/output device. From software's viewpoint, I/O devices exist as memory-mapped registers that are accessed in main-storage space by load/store instructions. The operating system typically configures access to I/O devices as caching-inhibited and guarded.
IOID	I/O Identifier. It is described in the IOPT and identifies an I/O unit.
IOIF	One of two I/O interfaces supported by the EIB.
IOIF device	A device that is connected to Cell Broadband Engine's IOIF port directly.
I/O operation	A storage operation that crosses a Cell Broadband Engine coherence-domain boundary.
IOIF protocol	The EIB's noncoherent protocol for interconnection to I/O devices. See *BIF*.
I/O unit	One or more physical I/O device(s), I/O bridge(s), or other functional unit(s) attached to an IOIF, in which one value of the IOID described in the IOPT. A functional unit that can initiate I/O accesses.
IPC	Instructions Per Cycle.
IPI	Interprocessor Interrupt. In the CBE processor, a software interrupt written by a PPE thread or an SPU to one of the Interrupt Generation Port (IGP) registers.
IR	Instruction Relocate.
IS	Instruction Issue.
ISA	Instruction Set Architecture.
ISEG	instruction Segment Exception.
ISI	Instruction Storage Interrupt.
ISRC	Interrupt Source.
JTAG	Joint Test Action Group. A test-access port defined by the IEEE 1149 standard.

KB	Kilobyte.
L1	Level-1 cache memory. The closest cache to a processor, measured in access time.
L2	Level-2 cache memory. The second-closest cache to a processor, measured in access time. An L2 cache is typically larger than an L1 cache.
LA	An LS address of a DMA list. It is used as a parameter in an MFC command.
LBIST	Logic Built-In Self Test.
ld	Load doubleword instruction.
ldarx	Load doubleword and reserve x-form instruction.
LEAL	A DMA-list effective-address parameter in an MFC command.
livelock	A state in which one or more units in a processor element cannot make forward progress, such as in an endless loop of program execution. Compare *deadlock*. In a livelock, processing continues to take place. In a deadlock, no processing continues.
lmw	Load multiple word instruction.
LPAR	Logical Partition. A virtual machine managed by the hypervisor.
LPCR	Logical Partition Control Register.
LPID	Logical Partition ID.
LR	Link Register.
LS	See *local store*.
LRU	Least Recently Used.
LSA	Local Store Address.
LSB	Least-significant byte.
LSb	Least-significant bit.
LSCSA	Local Storage Context Save Area,
LSU	Load and Store Unit.
lswi	Load string word immediate instruction
lswx	Load string word indexed instruction
lwarx	Load word and reserve x-form instruction
main memory	See *main storage*.

main storage	(1) The effective-address (EA) space. It consists physically of real memory (whatever is external to the memory-interface controller, including both volatile and nonvolatile memory), SPU LSs, memory-mapped registers and arrays, memory-mapped I/O devices (all I/O is memory-mapped), and pages of virtual memory that reside on disk. It does not include caches or execution-unit register files. (2) The level of storage hierarchy in which all storage state is visible to all processors and mechanisms in the system. See *local storage*.
MBL	MIC Bus Logic.
memory channel	The external XDR DRAM memory and supporting logic associated with an Rambus Extreme Data Rate (XDR) I/O Cell (XIO). The CBE processor has two XDR DRAM memory channels.
memory-mapped	Mapped into the Cell Broadband Engine's addressable-memory space. Registers, SPE local stores (LSs), I/O devices, and other readable or writable storage can be memory-mapped. Privileged software does the mapping.
MIC	Memory Interface Controller. The Cell Broadband Engine's MIC supports two memory channels.
MFC	Memory Flow Controller. It is part of an SPE and provides two main functions: moves data via DMA between the SPE's Local Storage (LS) and main storage, and synchronizes the SPU with the rest of the processing units in the system.
MiClk	MIC Core Clock.
MMIO	Memory-Mapped Input/Output. See *memory-mapped*.
MMU	Memory Management Unit. A functional unit that translates between effective addresses (EAs) used by programs and real addresses (RAs) used by physical memory. The MMU also provides protection mechanisms and other functions.
MSB	Most-Significant Byte.
MSb	Most-Significant Bit.
MPI	Message Passing Interface. A standard for high-performance communication on massively parallel architectures and clustered distributed-memory systems.
MSR	Machine State Register.
mtmsr	Move to machine state register instruction.
mtspr	Move to special-purpose register instruction.

NClk	CBE Core Clock. The clock for the PPU and SPU processor elements. It is the highest-frequency CBE clock.
NCU	Non-Cacheable Unit.
page table	A table that maps virtual addresses (VAs) to real addresses (RA) and contains related protection parameters and other information about memory locations.
PC	Program Counter.
PCAL	Periodic Calibration of DRAM timing.
pervasive logic	Logic that provides power management, thermal management, clock control, software-performance monitoring, trace analysis, RAS, JTAG, and so forth.
page table	A contiguous sequence of real pages beginning at the real address specified by SDR1 contains the Page Table.
PG	Processor-Unit Group.
PIR	Processor Identification Register.
PLG	Physical Layer Group.
PLL	Phase-Locked Loop.
POR	Power-On-Reset for the CBE processor.
PowerPC	Of or relating to the PowerPC Architecture or the microprocessors that implement this architecture.
PowerPC Architecture	A computer architecture that is based on the third generation of RISC processors. The PowerPC architecture was developed jointly by Apple, Motorola, and IBM.
PPC	PowerPC Core. An early name (now unused) for the PowerPC Processor Element (PPE).
PPE	PowerPC Processor Element. The general-purpose processor in the Cell Broadband Engine. It consists of the PPU and the PPSS.
PPU	PowerPC Processor Unit. The part of the PPE that includes execution units, memory-management unit, and L1 cache.
PPSS	PowerPC Processor Storage Subsystem (L2 cache, NCU, CIU, BIU). Part of the PPE. It operates at half the frequency of the PPU.
privileged mode	Also known as supervisor mode. The permission level of operating system instructions. The instructions are described in the PowerPC Architecture Book III and are required of software the accesses system-critical resources.

problem state	The permission level of user instructions. The instructions are described in the PowerPC Architecture Books I and II and are required of software that implements application programs. Compare *supervisor state*.
PTE	Page Table Entry. See *page table*.
PU	Processing Unit. An early name (now unused) for the PPU.
PVR	Processor Version Register.
QoS	Quality of Service. It usually relates to a guarantee of minimum bandwidth for streaming applications.
quadword	16 bytes.
RA	Real Address.
RAG	Resource Allocation Group.
RAID	Resource Allocation ID.
RAM	Resource Allocation Management. A mechanism that allocates access to Resource Allocation Groups (RAGs). Examples are the allocation of access to memory banks or I/O interfaces.
RAS	Reliability, Availability, and Serviceability unit. Part of the *Pervasive Unit* or *Pervasive Logic*. Also called the *TCU (*Test Control Unit).
real address	The set of all addressable bytes in physical memory and on devices whose physical addresses have been mapped to the RA space—such as an SPE's on-chip LS or an I/O device's off-chip register or queue.
RESN	Returned Envelope Sequence Number. In BIC. Upon successful reception of an envelope a positive acknowledge is generated back to the transmitting chip.
RLM	Random Logic Macro.
replacement management	The software management of cache-line and TLB-entry replacement, in order to increase cache-hit and TLB-hit ratios.
RMT	Replacement Management Table. Used by privileged software to control cache-replacement policy.
RO Clk	FlexIO Receive Clock.
RPN	Real Page Number.
RRAC	Redwood Rambus Access Cell, properly named Rambus® FlexIO™ processor bus. See *FlexIO*.
SBI	Synergistic Bus Interface. The MFC's interface to the EIB.
scarfing	The direct transfer of data to the PPE L2 cache.
SDR	Storage Descriptor Register.

SG	SPU Group.
SLB	Segment Lookaside Buffer. It is used to map an effective address (EA) to a virtual address (VA).
SLIH	Second-Level Interrupt Handler.
SMP	Symmetric Multiprocessor.
snoop	To compare an address on a bus with a tag in a cache, in order to detect operations that violate memory coherency. Also called *inquire*.
SPE	Synergistic Processing Element. It includes an SPU, an MFC and an LS.
SPI	Serial Peripheral Interface. A serial bus that connects the CBE Pervasive Logic to an external System Controller.
SPR	Special Purpose Register.
SPU	Synergistic Processor Unit. The part of an SPE that executes instructions from its local store (LS).
SRR0/SRR1	Save and Restore Registers 0 and 1.
starvation	A condition in which a processing element is making forward progress, but at an extremely slow rate.
static	A signal is said to be static if it is tied high or low.
sticky bit	A bit that is set by hardware and remains so until cleared by software.
supervisor mode	See *privileged mode*.
TB&D	Time Base and Decrementer.
TCU	Test Control Unit. Part of the *Pervasive Unit* or *Pervasive Logic*. Also called the *RAS* (Reliability, Availability, and Serviceability) unit.
thread	A sequence of instructions executed within the global context (shared memory space and other global resources) of a process that has created (spawned) the thread. Multiple threads (including multiple instances of the same sequence of instructions) can run simultaneously, if each thread has its own architectural state (registers, program counter, flags, and other program-visible state). Each SPE can support only a single thread at any one time. The multiple SPEs can simultaneously support multiple threads. The PPE supports two threads at any one time, without the need for software to create the threads. The PPE does this by duplicating architectural state.
time base	The CBE-processor facility that provides the timing functions for the CBE Core-Clock (`NClk`) domain.
TIS	Tool Interface Standard.

TKM	Token Management Unit. Part of the Element Interconnect Bus (EIB) that software can program to regulate the rate at which particular devices are allowed to make EIB command requests.
TLB	Translation Lookaside Buffer. An on-chip cache that translates virtual addresses (VAs) to real addresses (RAs). A TLB caches page-table entries for the most recently accessed pages, thereby eliminating the necessity to access the page table from memory during load/store operations.
TO Clk	FlexIO Transmit Clock.
training	Same as *calibration.*
VA	Virtual Address.
Vector/SIMD Multimedia Extension	The SIMD instruction set of the PowerPC Architecture, supported on the PPE. Also known as AltiVec, which is a Motorola trademark.
VID	Voltage ID.
virtual address	An address to the virtual-memory space, which is much larger than the physical address space and includes pages stored on disk. It is translated from an effective address (EA) by a segmentation mechanism and used by the paging mechanism to obtain the real address (RA). The maximum size of the virtual-address space is 2^{65} bytes.
virtual memory	The address space created using the memory management facilities of a processor.
virtual mode	The mode in which virtual-address translation is enabled.
VPN	Virtual Page Number. The number of a page in virtual memory.
VRM	Voltage Regulator Module.
VSID	Virtual Segment ID.
VSU	Vector Scalar Unit. In the PPE, the combination of the VXU and FPU.
VXU	Vector/SIMD Multimedia Extension Unit.
WIMG bits	Four bits in a page table which control the processor's accesses to cache and main storage—“W” is write-through, “I” is caching-inhibited, “M” is memory-coherence, and “G” is guarded.
word	Four bytes.
XDR	Rambus® XDR DRAM technology.
XDRIG	The *Rambus XDR Initialization Guide (DL-0178)* supplied by Rambus.
XIO	Rambus® XDR I/O (XIO) cell. See *XDR*.

YRAC	Yellowstone Rambus Access Cell, properly named Rambus® XDR DRAM cell. See *XDR* and *XIO*.

Revision Log

Revision Date	Version	Contents of Modification
November 8, 2005	0.1	Draft A of the following chapters: • Signal Descriptions • Serial Peripheral Interface (SPI) • Configuration Ring
November 29, 2005	0.2	Draft A of the following chapters: • Initialization Sequence
December 16, 2005	0.3	Draft A of the following chapters: • Preface • Introduction • MMIO Registers • FIR Overview • Livelock Resolution Mode • Sample XDR DRAM Memory Calibration Pattern • Glossary
February 28, 2006	0.9	Pre-Final Draft of all chapters.
March 13, 2006	1.0	Final Draft of all chapters.
March 20, 2006	1.1	Corrected Table D-1 and related text.
March 30, 2006	1.2	Removed confidentiality notices from headers and footers, modified inside front cover statements, and removed a few statements about procuring confidential documents under non-disclosure.
March 31, 2006	1.3	Removed two sentences from inside front cover statements.